Broadview
www.broadview.com.cn

大产品，小团队

携程敏捷技术与管理转型实战

携程技术中心／编著

由初心到零字，创造无限可能，
18岁青春撰程成人礼

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

作者简介

作为携程旅行网的核心竞争力，由超过3000 位来自海内外的精英工程师组成，为携程旅行网业务的运作和开拓提供全面技术支持。以技术创新为产品、服务创造价值，以技术引领业务增长，使携程技术中心成为更优秀的 OTA 技术团队。

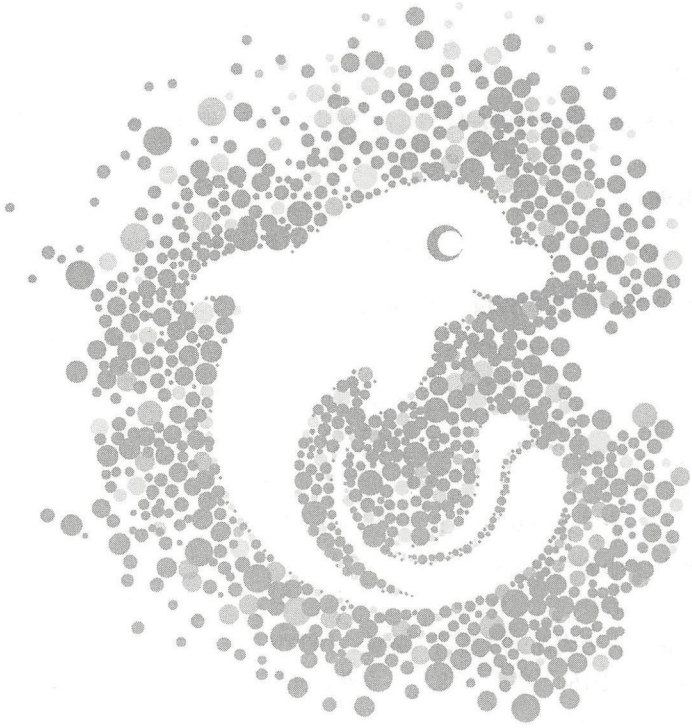
携程技术中心

我们拥有 10 年以上互联网产品项目管理经验。我们乐于分享，自发组织了一个小团队，把我们在产品、项目、技术等专业领域的收获贡献给敏捷社区。为此，我们集结了 10 位总监和 20 位一线专家分享亲身实战经验，还特别邀请了火车票事业部和创新工场 CEO 陈刚以及车船票和租车事业部 CEO 王玉琛亲自执笔，带你修炼绝世武功。

大产品，小团队

携程敏捷技术与管理转型实战

携程技术中心／编著



電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

敏捷并不是什么新玩意儿，但它已经成为这个瞬息万变的互联网+科技商业时代的主流管理运营体系。如果一个企业还没开始拥抱敏捷并付诸实践，那它很快就要被淘汰了！

现在我们遇到的问题大多是如何让敏捷落地，如何把敏捷带给整个企业。本书并不是敏捷方法教授的纯理论书，作者只是把 5 年敏捷转型中趟过的那些坑，吃过的那些亏，流过的那些泪……通过一个个鲜活的案例呈现出来，送给那些已经开始尝试敏捷但可能遇到一些问题的人，以及虽然没有开始但已经跃跃欲试准备实践敏捷的人。

本书适合初级和资深 Scrum Master、产品经理、技术管理者、项目经理，以及敏捷爱好者学习和参考。

图书在版编目（CIP）数据

大产品，小团队：携程敏捷技术与管理转型实战/携程技术中心编著. —北京：电子工业出版社，2018.1
ISBN 978-7-121-32903-6

Ⅰ. ①大… Ⅱ. ①携… Ⅲ. ①电子商务－企业管理－产品管理 Ⅳ. ①F713.36

中国版本图书馆 CIP 数据核字(2017)第 259736 号

策划编辑：孔祥飞
责任编辑：徐津平
印　　刷：三河市华成印务有限公司
装　　订：三河市华成印务有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编：100036
开　　本：720×1000　1/16　印张：18.5　字数：361 千字
版　　次：2018 年 1 月第 1 版
印　　次：2018 年 1 月第 1 次印刷
定　　价：59.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819，faq@phei.com.cn。

推荐序

携程自 1999 年成立以来，到现在已经走过了 18 个年头，携程的成长伴随并见证了中国旅游行业发展的每个关键时期。从大规模旅游服务呼叫中心时代，到互联网时代，再到当今移动设备普及。现在大数据和智能化驱动行业的发展更加快速，携程也在不断探索和创新，为旅游用户带来价值。在这 18 年的探索中，担负着“让旅游更幸福”的使命，不断完善产品和服务的细节并努力将其做到极致。产品的背后是艰苦的工作，是多团队协同作战以及不断快速迭代版本，而科学的项目管理方法至关重要。

互联网产品的一个重要特点是快，对用户的需求变化需要快速响应，对产品上线后的效果需要快速验证，版本需要快速迭代，在多团队协调作战的情况下，如何在确保质量的前提下做到快，我们鼓励每个团队不断探索自己的最佳实践。公司有统一的研发流程来确保不出现灾难性的后果，但是无法为每个团队都带来最好的效率，不同的团队根据自己的具体情况做出调整，并将好的实践经验反馈回统一研发流程中，在不断的实践探索中，我们的项目研发效率也不断提升，项目交付质量也持续维持在很高的标准。

我鼓励项目管理的同事们将我们这些年探索敏捷开发的经验和教训记录下来，同时分享给所有人，这既是我们的故事，记载着我们的青春，同时也是携程步入 18 岁的成年礼。在新的全球化征程中，我们将继续探索，为了让全世界的用户在旅途中更幸福而努力奋斗。

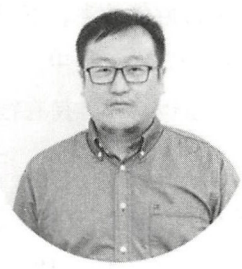
携程旅行网集团 CTO　甘泉

作者团队介绍

曾经，我们也是小白，只是我们有属于自己的梦想，愿意挑战自我，去探索未知的世界。

现在，我们拥有 10 年以上互联网产品项目管理经验。我们乐于分享，自发组织了一个小团队，把我们在产品、项目、技术等专业领域的收获贡献给敏捷社区。为此，我们集结了 10 位总监和 20 位一线专家分享亲身实战经验，还特别邀请了火车票事业部和创新工场 CEO 陈刚以及车船票和租车事业部 CEO 王玉琛亲自执笔，带你修炼“绝世武功”。

未来，希望我们所在的团队——携程技术中心，持续成长为世界顶尖的 OTA 技术团队。



王凯——携程技术中心 PMO 负责人；携程大学首席项目管理讲师；中国十大优秀项目管理培训师；CSM，CSPO，PMP

15 年以上项目管理从业经验，致力于研究“中国式项目管理方法”。曾在通信行业从事项目管理 10 余年，在复杂组织结构下的跨地区项目管理方面具有很好的实践经验。

目前效力于携程旅行网 5 年多，经历了技术研发中心从 300 人到 3000 人的过程，主导和参与过很多跨组织项目。如无线技术部拆分为各事业部融合项目、香港永安融合项目、台湾易游网融合项目、途风网技术合作、去哪儿网联合项目、艺龙旅行网联合项目……个人特别喜欢尝试不同有挑战的工作，尤其是组织变革带来的新机遇和未知的风险，会有一种莫名的快感。

《大产品，小团队：携程敏捷技术与管理转型实战》 主审

黎娟——去哪儿网过程改进总监

15 年以上软件项目管理及过程改进经验。曾先后就职于雅虎中国、阿里巴巴、腾讯、去哪儿网。负责组建项目经理团队、建设产品研发管理体系、主持大型研发项目，并推动整体产品研发流程优化升级。擅长问题分析以及基于问题驱动的过程改进，致力于建设高效且有幸福感的团队。

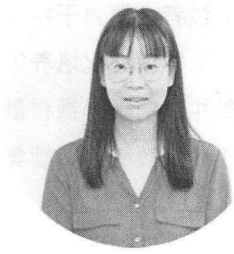
《大产品，小团队：携程敏捷技术与管理转型实战》 主审

季国红——携程资深敏捷教练；CSM，CSP，PMP

10 年以上互联网产品项目管理经验。曾先后就职于盛大游戏、携程旅行网，负责公司战略级项目实施及研发管理体系建设。

致力于企业级大规模敏捷转型，Scrum/XP 实施，助力研发效能提升。

本书主编及“第 1 章 新东西，一个思考”出品人



马力——携程资深项目经理及 Scrum Master；CSM，PMP

从 2015 年年初至今，负责酒店无线 5 个团队的敏捷转型。致力于提高研发效率以及团队建设，引导团队采用 Scrum 框架。和一线团队一起工作，共同探讨敏捷多种模式，落地实践，持续改进，追求一种幸福的工作模式。

“第 2 章 如何组建高效的小团队”出品人

马赛珺——携程 Scrum Master；CSM

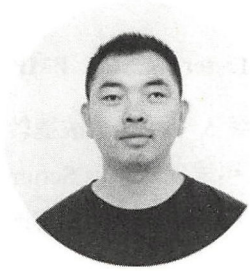
携程机票前台 Scrum Master，机票敏捷转型的主要负责人。致力于帮助团队使用 Scrum，并持续改进，打造高效团队。机票已由初始的 5 个敏捷团队发展为 12 个 Scrum 团队。曾带领 2 个团队参赛并成功获得携程最佳敏捷团队奖。

“第 3 章 打造幸福的小团队”出品人

杨春勤——携程高级项目经理

负责公司内黄埔训练营，致力于通过 OK 制和 OKR 的培训传播、释放团队创业激情，提升产品研发效率。还先后负责过携程 APP 项目集管理和机票事业部前端一百多人研发团队的敏捷转型。加入携程前，经历明基逐鹿软件 BPM 产品研发管理和大智慧 DTS 项目集管理。

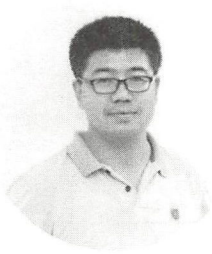
“第 4 章 极致敏捷的小团队”出品人



罗勇——携程云平台高级研发经理

携程早期敏捷、精益方法布道者，实践者。致力于持续交付工具链打造、端到端研发效率提升、工程师文化培养等方面的研究。曾带领云平台研发团队参赛并成功获得携程最佳敏捷团队奖。曾经就职于爱立信，担任敏捷教练、持续集成平台架构师等职务。

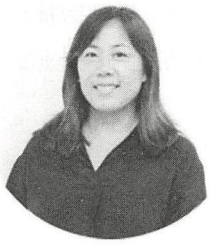
“第 5 章 效率为王，唯快不破”出品人



韩少勇——携程攻略社区资深项目经理；CSM，信息系统项目管理师，数据经理

先后经历携程攻略社区 Web 网站、携程 APP 攻略及行程模块、携程攻略独立 APP 等产品的项目管理工作。曾供职于摩托罗拉，专注于流程优化、数据分析、互联网产品价值链的改善。

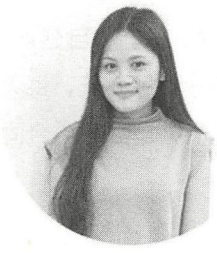
“第 6 章 用敏捷思维做大产品”出品人之一



宋萍——携程民宿高级产品经理；CSM，PMP

负责民宿平台搜索排序、分销等业务。2016 年带领民宿、团购团队进行敏捷转型，尝试 Kanban、Scrum 等多种敏捷实践。曾就职于三星电子，担任研发经理、项目管理等职务，专注于产品交付。

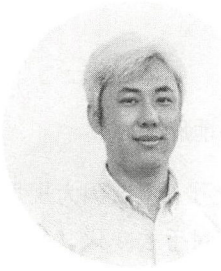
“第 6 章 用敏捷思维做大产品”出品人之一



贾露——携程产品经理；CSPO，PMP

2014 年带领度假搜索团队敏捷转型，获得过携程首届 Scrum Master 最具价值专家奖（MVP）。担任过搜索算法排序、推荐 PO，致力于产品价值交付。

“第 6 章 用敏捷思维做大产品”出品人之一



陈强——携程资深经理；CMMI，PMP/ACP；信息系统项目管理师；CSM/CSPO

研发过程改进及变革导入者。敏捷教练，项目管理“本本族”。

“第 6 章 用敏捷思维做大产品”出品人之一

前　言

敏捷，是一个很有吸引力的话题，近几年也非常火，已席卷全球。即使敏捷已经成为主流，一个企业也不应该为了敏捷而敏捷，敏捷是要解决具体问题的。企业的具体问题在哪？有哪些痛点？涉及哪些方面？这些问题要好好梳理。

那些年我们踩过的坑

2011 年，我加盟了携程旅行网，自恃在项目管理圈混迹了 10 年有余，积累了一些经验与实用的方法，可以快速复制，帮助提升公司的项目管理能力。谁知，事与愿违，当时携程的项目管理环境和之前作为乙方公司的项目管理环境完全不同。

乙方公司对项目的范围、进度，尤其是成本有着相当严格的要求，一旦项目延期，就会给公司的收益造成很大压力。因此，项目经理通常会把计划细分到每一天，每天来回顾当天的任务是否完成，每周都会与客户沟通当周的任务完成情况是否符合合同要求，一旦发生偏差，会立马纠正并通过加班加点赶上计划。

那时候，携程的项目管理是自己做自己的项目，没有甲乙方的概念，大部分项目对交付时间都没有强制的要求，导致进度变更的项目比例都在 50%以上，其中有最主要的两个原因：

- 有优先级更高的项目插队进来，占用原来项目的资源。
- 规划不充分（需求不清楚），导致投入了很多无用功，不停地反复。

除了进度变更，还有一些项目甚至被取消或者做到一半被中止了，主要因为规划周期过长（提前半年规划项目），有些时候因为市场的变化和竞争对手的变化，已经

没有必要再做了或者有更重要的项目需要去做。

当时携程的研发人员有 300 多人，项目经理与研发人员的比例是 1:10，大约 30 人左右，人均年项目在 3~4 个，项目的平均周期都要 6 个月以上。由于一半以上的项目都未能正常地按计划完成，项目经理的挫败感还是挺严重的，在他们看来，资源始终是紧张的，计划永远是赶不上变化的，到头来，真正按计划保质保量完成的项目，更是屈指可数的。

携程的 PMO（项目管理办公室）也在那个时候设立，主要的职能就是帮助整个研发中心建立项目管理流程，开发适合于携程这个组织环境下使用的管理工具（包括项目管理工具），组织一些初级的项目经理进行培训，提升他们的能力。当时很多项目经理都是研发人员转岗而来的，也有一些是通过外部招聘而来的，这就形成了新旧两派人员，他们的作风和思想形成了非常明显的对比。“老人”们对原来的流程已经适应了，对于频繁的变更和提前很长时间规划项目已经习以为常，觉得存在即合理；“新人”们则比较不适应这种环境，他们当中大多来自作为乙方的公司、互联网企业和外企。我作为当时的“新人”，对于这种项目管理方式自然是嗤之以鼻的，甚至一度认为这种管理方式“很 low”。

不畏转型，坚定前行

现今回头来看，当时那样的环境是有它存在的道理的。2011 年以前，携程在国内 OTA 行业已经雄霸多年，当时主要的收入都来自于 Offline 人工客服，线上 PC 端的份额仅为 25%左右，无线业务刚刚开始起步，竞争对手逼得不是很紧，很多线上和无线端的项目属于试水阶段，往往一些项目早早规划好了，会因为市场的变化而做出调整，项目变更频繁也就在情理之中了。

从 2012 年开始，OTA 市场竞争愈来愈激烈，竞争对手开始大力发展无线端业务，把用户往手机端引导，产品品类也逐步增加，不仅仅是我们传统意义上的机票、酒店、度假产品预订了。这就要求项目上线速度非常快，之前动辄半年以上的项目周期显然已经不能满足当下的市场变化了，一个新产品功能的上线，往往不能超过 2 个月，短则 1 个月内就需要上线，因为晚了，可能就被对手抢占了先机。这个时候，项目经理中的“新人”提出了一个想法：我们何不试试看用敏捷的方法去做项目。于是，就有了第一支敏捷实践的团队，那就是“舌尖上”的 Scrum 团队，其中的艰辛就不在这里

多说了，本书在后面的章节中会提到该团队的成长史。

但凡在组织内推行过方法的人都知道，一种新的理念与方法，要让组织内的成员接受，首先你自己得实践成功，其次你得说服老板支持你，更重要的是，有些时候必须调整组织架构才能真正推行下去。携程的敏捷项目管理推行的过程就是一部真实的血泪史，团队经过了漫长的 5 年时间，终于修成正果，完成了一个小目标，那就是整个研发团队的敏捷方法覆盖率达到了 80%。

我们只讲实战不谈理论

本书并不是敏捷方法教授的纯理论书，我们只是把 5 年敏捷转型中趟过的那些坑，吃过的那些亏，流过的那些泪……通过一个个鲜活的案例呈现出来，让正在进行敏捷转型的公司，或是与我们一样，即将转型成功的公司作为一个参考，借此和同行们交流分享，不尽之处请大家帮忙指正。

本书内容分为四大部分。

NO.1 理念篇

敏捷虽然不是什么新鲜玩意儿，但在每个组织落地过程中，会遇到各种各样的问题。希望读者能够对以下问题进行思考：我们为什么要敏捷？敏捷能给我们带来什么？如何变得更加敏捷？这里，我们运用多种思维模式对这些问题进行回答，最终，我们将会得到全方位的敏捷世界。

NO.2 团队篇

将亲身实践作为出发点，结合自身痛点共同探讨：如何组建高效的小团队，打造幸福的小团队以及成为极致敏捷团队的方法。本篇将让你更加清晰地认识和探索充实的敏捷之旅，持续收获幸福和成长。

NO.3 技术篇

打造高效的敏捷团队，不仅和组织架构、研发流程、团队氛围有关，还需要强大的工具支持，才能切实提升团队的交付能力。在这里，你可以找到覆盖产品生命周期

全链路的各种工具，看它们是如何帮助我们提升技术效率的，希望同样可以帮助到你。

NO.4 产品篇

如何善用敏捷开发快速迭代和验证产品的价值？如何应对自上而下、打配合战、自下而上的不同类型的产品？如何主导推动大型产品运作？如何通过创新驱动产品迭代升级？在本篇中，或许可以给产品经理们一些启发。

读完本书后，如果感觉内容有用，您就给我们点个赞。我们的故事都是真实的，通过这些案例，希望让您感受到组织的变革、管理方法和工具的变革是多么不易，这使我们受益良多。谨以此文与各位共勉！

编者　王凯

敏捷增值服务介绍

携程技术中心 PMO 团队除了服务携程集团，对外也提供如下公众服务：

1．敏捷转型辅导与咨询服务；

2．敏捷项目管理讲座（敏捷、项目管理、产品创新等）；

3．敏捷转型辅导与咨询服务；

4．携程相关产品合作机会；

如果您有兴趣，欢迎与我们联系和咨询。

邮箱：EPG@Ctrip.com。

我们将于每年 5 月 20 日举行敏捷总动员活动， 并不定期传递敏捷实战故事、模板工具、业界资讯等信息。

【敏捷总动员】微信公众号：

读者服务

轻松注册成为博文视点社区用户（www.broadview.com.cn），扫码直达本书页面。

- **提交勘误**：您对书中内容的修改意见可在 提交勘误 处提交，若被采纳，将获赠博文视点社区积分（在您购买电子书时，积分可用来抵扣相应金额）。
- **交流互动**：在页面下方 读者评论 处留下您的疑问或观点，与我们和其他读者一同学习交流。

页面入口：http://www.broadview.com.cn/32903

目　录

NO.1　理念篇

NO.2　团队篇

NO.1
理念篇

敏捷虽然不是什么新玩意儿，但在每个组织落地过程中，会遇到各种各样的问题。希望读者能够对以下问题进行思考：我们为什么要敏捷？敏捷能给我们带来什么？如何变得更加敏捷？这里，我们运用多种思维模式对这些问题进行回答，最终，我们将会将会得到全方位的敏捷世界。

第 1 章

新东西，一个思考

我们身处瞬息万变的互联网+科技商业新时代，各大互联网公司规模庞大，产品更新迭代日新月异。那么企业如何永葆活力，保持团队迅速的反应能力，不断创新呢？

1.1 初识敏捷，从零开始

俗话说，船小好掉头。我们的经验就是，业务线解耦，把大团队变成很多小团队，这样团队战斗力更强，反应更“敏捷”。当我们敏捷转型时，会面临各种各样的问题，我们需要好好思考，还要有勇气去突破。

问：如何按业务线划分团队？

答：从业务角度看分为售前、售中、售后；从客户角度看分为行前、行中、行后。团队的组建要从客户角度出发进行划分。

问：一个大的团队如何拆分为 10 人的小团队呢？

答：这是高层管理者需要思考的问题，要组织边界划分，考虑组织架构，降低复杂程度。拆分后，我们的组织只包含 CEO、VP、总监、Team Leader 这四个层级。

京东的团队也有“8150”原则，创始人刘强东认为，这是京东扁平化管理 13 万名员工的秘诀：“从我到基层只有 4~5 层，这个原则保证我和基层员工的距

离尽可能短。”

第一个数字是“8”，即向一个管理者直接汇报的下属不得低于 8 人。在京东，不存在副总裁管两个总监的情况。如果不够就合并，减少管理职位，否则管得太细。

第二个数字是“15”，即原则上管理人数不超过 15 人。如果超过 15 人，管理者就会疲于奔命。只有超过 15 人才可以增加一个层级。

第三个数字是“50”，单一工种的基层管理者（如分拣、打包）至少管理 50 人，超过 50 人才可以分出两个主管。

问：原来是职能化团队，转变为敏捷团队后，最大的问题是人力问题，人力资源是更有效还是会有一定的浪费，还是会带来其他好处呢？

答：人力问题在敏捷组织架构下对 Team Leader 和 PM 的要求会更高，人力资源由他们来定，相比职能化组织，敏捷团队人力有冗余，人员会跨职能，需要培养。冗余问题，团队自己做保留动作，资深程度也由自己决定。

问：划分后，在技术的提升上可能会变慢，提升空间会减小么？

答：技术提升的空间存在，团队人员技能提升到什么程度取决于领军人物的高度。技术团队要多交流分享，可以有虚拟的社区运作。

问：新产品怎么推进？

答：在敏捷组织中，PO 来驱动项目，项目经理的角色要有所改变，有个快速适应的过程，需要跨团队工作，根据业务的发展解散或者动态生成新产品团队。

问：系统解耦和项目分解，组与组之间如何有效协作？

答：业务研发目标一致，团队开放、合作，信息透明、共享，Scrum Master 和团队主动推进跨团队工作。

搞清楚了敏捷转型过程中碰到的这些问题，如果答案是能够解决的，那么可以跟我们一起来看看敏捷是什么？敏捷能为我们带来什么？

1.1.1　敏捷是什么

提到敏捷，可能大多数人想到的是小团队、快节奏、增量开发、迭代测试、迭代发布等。除此之外，我个人对于敏捷的理解是：其核心点不仅仅是研发速度上的快，

更重要的是整个团队通过最短路径达成业务目标。

团队通过 Scrum 的模式来实施敏捷研发，PO 主导着产品项目的方向，Scrum Master 把控流程和进度，团队成员负责开发、测试。作为 Scrum 团队成员，规范上大家各司其职，井然有序的工作，文化上大家需要像橄榄球队员一样具备充分的主人翁意识，“你争我夺”地共同完成任务目标。

我认为：PO 不仅扮演着产品经理的角色，定义需求和研发交付物，而且需要与团队共同制订需要达成的业务目标，这个目标不仅靠产品的设计来达成，而且靠整个团队共同实现。

那么，何为“通过最短路径达成业务目标”呢？那就是，业务目标制订→产品定义设计→开发编码→测试→上线整个流程和内部的迭代，要在整个链条中不走或者尽可能少走冤枉路。所以，敏捷的含义并不单纯是实现快速研发，而是快速达成业务目标。为保证这一点，在各个环节都需要尽可能避免做无用功，例如 PO 在业务目标决策制订上合理，产品功能、交互设计上明确，与研发团队沟通充分避免信息不对称，全程质量保证，尽可能在项目早期暴露缺陷等。

举几个反面案例进行探讨：

A. 领导下达指令要求业务指标需要提高到 XX，PO 未做详细调研和推算，准备先做做看，于是制订产品改进方案，很快做出 PRD 并通过开发团队评审。Scrum Master 跟进协调资源，开发人员和测试人员认领任务后，快速完成任务并且顺利发布上线。经过一段时间观察，业务指标并未达到领导预期，于是 PO 再出改进方案，但收效仍然达不到领导的要求。

B. PO 根据 OKR 分解，制订了一系列的业务目标并经过评估基本可达成，产品经理给出产品设计后为了快速抢占市场验证效果，研发团队加班加点完成功能的实现，发布上线并开始 AB test。但是部分指标离预期差距较大，产品经理和团队成员猜测可能受某些页面交互的影响，于是快速改进并上线验证，发现有效果但仍然并不显著，计划再进行改进但苦于无详细数据作为方向指导，版本始终陷入不断试错并改善缓慢的境地。

C. 团队研发系统改版增强全方面用户体验，预计上线后将显著提升用户的转化率。研发团队拆解任务并按照 Scrum 模式进行研发，完成功能并对用户关键路径进行埋点，将系统快速推上线。进行灰度发布后开放部分流量进行观察，一段时间后发现某种场景下存在较严重的 Bug 并影响转化率，同时发现埋点存在缺失，无法准确记录用户的行为数据。于是 PO 紧急关闭流量，讨论

如何修复功能和埋点。

以上几个案例都假设了研发过程中采用了所谓的敏捷方式，能够快速地将产品设计的功能实现并发布上线，但是我们需要思考，从整体看我们真的敏捷起来了吗？三个案例在研发层面上都尽可能保证快速上线，但是在项目的整体效果上未达成目标，导致在不同环节进行返工，即文中提到的在项目整个链条中走了一定的冤枉路，如果这种情况造成较大的资源损耗，那么我们的项目无法被称为成功地实施了敏捷管理。

案例 A 中，上级领导提出了对某些业务指标的期望，作为 PO 应该对此做一定的评估，确保我们的业务目标是能够达成的。如果在依照目前的现实情况下无法达成，或者需要非常大的成本代价并经过评估团队目前是无法承担的，那么需要 PO 勇于基于数据和分析结果向上级领导表达观点。在 Scrum 团队也需要具备这样的文化，否则整个 Scrum 团队可能陷入在死胡同里转圈的局面。

案例 B 中，整个团队在规划和实施产品功能方面可能没有太大的问题，但是在实施过程中缺少对用于业务关键指标分析的辅助数据的收集和监控，这会造成产品经理在优化产品的过程中更多地靠猜测，而没有科学客观的数据支撑，陷入了产品“我以为……”的想当然情况，也就产生了很多被称为“自嗨”的产品功能。因此，详尽的路径和行为埋点更有利于产品经理做出更准确的决策，在优化产品时不带着整个 Scrum 团队“绕圈”，否则即使产品上线速度再快也无法称之为真正的敏捷。

案例 C 则显然是产品功能和相应的埋点的质量没有做到很好的保证，导致上线后又被迫关闭流量进行返工修复。软件测试的方法论中提到缺陷越晚暴露则修复成本越高，这里提到的成本不仅仅是人力资源和时间的成本，对于敏捷团队来说也会造成对团队响应能力的负面影响。敏捷强调团队对于产品业务目标达成统一关注和快速、机动应对，一个疲于奔命于修复线上 Bug 的敏捷团队，如何能够敏捷地应对业务带来的变化呢？！

1.1.2　Kanban 是什么

Kanban 的目标是促进工作在整个价值链中快速流动。使用 Kanban，对于组织来说，不需要新的流程，不需要新的角色，更不需要恼人的组织架构重组。

为什么使用 Kanban

- 计划赶不上变化
- 估算不准，交付总是延迟
- 工作被打断，团队处于救火状态
- 组织架构动不了

Kanban 的基本原则

可视化：创建代表工作项的便利贴，在工作流白板上按列展示并跟踪每个工作项，这样可以了解团队的工作，反思工作是如何运作的，并发现在工作流程中的改进机会。

限制 WIP：针对团队能够同时进行的工作项人为地设置限制。这样做最明显的好处就是同时处理较少的工作项，使得每项工作更快地完成。

管理流：管理工作流使之快速且毫无中断地流动起来。流程中如果存在某个瓶颈拖延工作，在白板上很容易显现出来，往往越严重的问题越早暴露，而且一旦解决掉，工作的流动情况就会明显改善。

我们的关注点

- 关注端到端交付

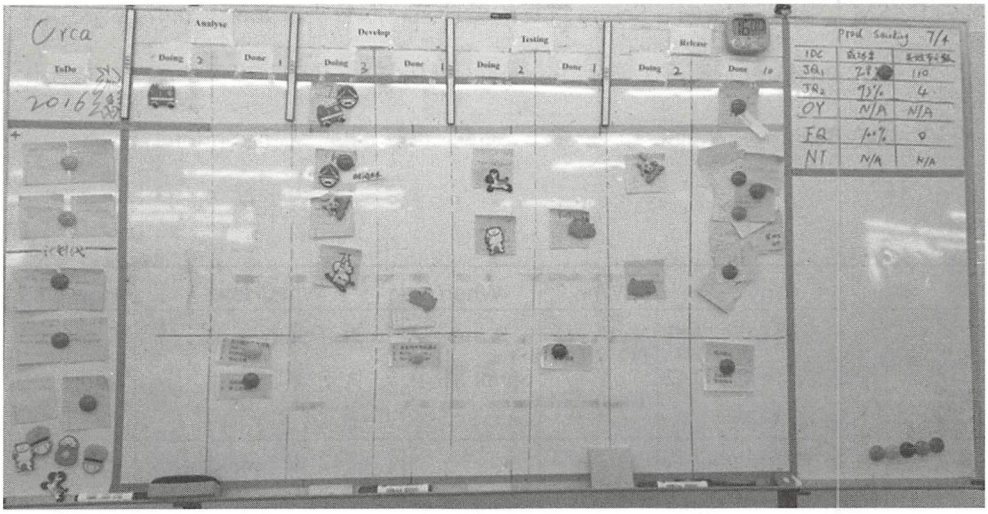
从需求分析到最终功能上线，并获得用户的正向反馈。

- 管理价值流动

消除瓶颈，确保每周有三个左右的功能点发布生产。

- 反应更敏捷

即时响应线上问题，让最合适的人直接处理问题。

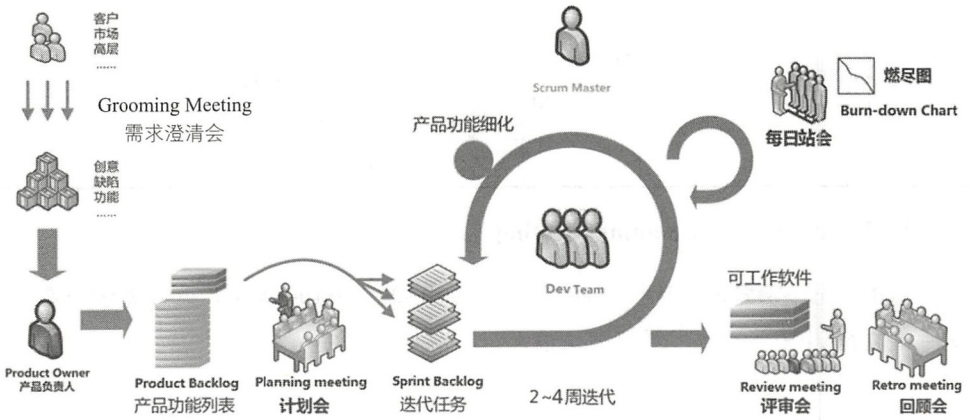


团队成员每日任务一目了然

1.1.3　玩转 Scrum

如果你对敏捷开发感兴趣，那应该听过 Scrum。Scrum 使用迭代的开发方式，每一次迭代，都会经历一个“计划→实施→验证→反思”的过程。具体是怎么运作的呢？跟我来一步一步轻松了解 Scrum 吧。

Scrum 框架包括 3 个角色，5 个会议，3 套工具。Scrum 框架流程如下图所示。



3 个角色

1．SM：Scrum Master，Scrum 过程的管理者，服务于 PO、团队和组织。

2．PO：Product Owner，对产品 Roadmap 和 Backlog 负责，确保产品价值最大化。

3．Dev Team：架构师、开发人员、测试人员等，负责实现 Sprint 目标。

5 个会议

	Why	Who	When	What/How	How long
需求澄清会	把不清楚的需求梳理清楚，为下面 1~2 个 Sprint 准备	PO、Dev Team、SM	Sprint 期间	1. 拆分 Story 2. 优先级排序 3. 澄清	2 小时
计划会	把清楚的 Product Backlog 变成 Sprint Backlog，确定 Sprint 交付增量以及如何完成	PO、Dev Team、SM	Sprint 开始前 回顾会之后	1. PO 讲解 Sprint 目标及待办列表 2. Team 预测 Sprint 开发功能 3. Team 确定如何完成	2 小时
每日站会	为了开发活动同步制订下一个 24 小时计划	Dev Team、SM	每天	1. 检视昨天 2. 计划今天 3. 确认和清除障碍	15 分钟内
评审会	检视，调整	PO、Dev Team、SM	Sprint 结束前	1. Demo 2. 收集反馈 3. Review DoD（众测）	1 小时
回顾会	检视、改进	Dev Team、SM	评审会与计划会之间	2. 检视：人、关系、过程、工具 2. 成就、困难/挑战、解决方案 3. 制定改进计划	1 小时

1．需求澄清会（Grooming Meeting）

目的：把不清楚的需求梳理清楚，为下面 1~2 个 Sprint 准备，拆分需求成最小可交付单位 MVP（可独立上线且有价值）。

活动：

1）PO：围绕需求背景、收益、价值、方案、交互这些要素介绍需求。

2）团队：理解需求、PK 需求、可行性评估、拆分 User Story。

小尝试：

1）怎么处理技术改进需求?

2）如果团队成员对所做的事情都不了解怎么办?

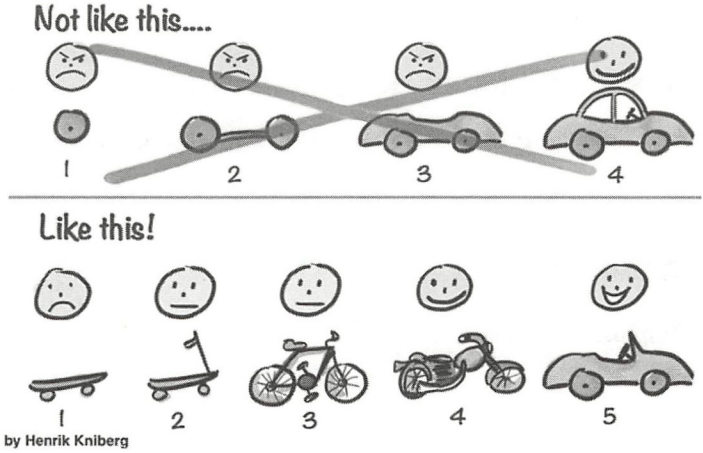
术语：

1）User Story：一个最小可独立上线且有价值的功能。

- 这个功能需要从用户的角度对系统的某个功能模块做出简短描述，以及了解完成这个功能之后将会产生什么效果，或者说能为客户创造什么价值。
- 通常情况下 1 个 Sprint（2 周 1 个迭代）可以做 4～5 个 Story，50 个左右的 Task（每个 Task 最好不要超过 8 小时，保证在 1 个工作日内完成）。
- User Story 用于描述用户故事，不要包括任何的技术、框架等内容。Task 可以包括技术、框架等内容。

2）MVP：Minimum Viable Product，最小可行产品。

- Not like this 代表了劣质的敏捷开发，从技术上看，它可能是增量和迭代交付的，但由于没有形成真正的反馈环，使得产品开发变得极具风险。
- Like this，聚焦于客户想要满足的潜在需求，找到最简单和最便宜的客户问题解决方法。如下图 Like this 所示，滑板是 MVP 产品，它比走路更快，用起来很简单，很多人都喜欢。接下来，你可以给滑板加个扶手，加个轮子，加个座位，加个齿轮，让用户可以用手扶着，坐下来，速度更快、更安全，最终成为一辆完善的汽车。



2．计划会（Planning Meeting）

目的：定出本次 Sprint 的目标和计划，让团队可以更顺畅地进行各自的工作。

活动：

1）团队在对故事点理解的基础上，对其进行估算 Story Point（一个相对独立的功能点，它能使不同技术水平和工作速度的人在估算结果上保持一致）。

2）事先调研方案，在计划会上对着代码及设计方案进行评估，开发人员围绕着设计方案进行讨论并且拆分任务，测试人员围绕着需求和设计方案产出测试用例。

小尝试：

1）PO 说没空参加怎么办?

2）怎么处理以前 Sprint 的 Bug?

3．每日站会（Daily Scrum Meeting）

目的：

1）加强团队交流和信息共享。互相了解彼此都在做什么工作，完成了什么任务。这样，每日的信息传递，可以让每个人更多地了解整个项目的业务和技术状况。并且如果在工作中遇到障碍或问题，也可以在这时候提出来，请求大家的帮助。

2）促使每个人在早上做好一天的工作计划。这样，每个人一天的工作就会有明确具体的目标，这会直接提高一天的工作效率。

活动：

在每日站会上，每个人只可以说三件事情：

1）我昨天做了什么？

2）我今天准备做什么？

3）我在工作中遇到了什么问题？

小尝试：

1）怎么控制站会在 15 分钟内完成？

2）站会下午开可以吗？

3）PO/Scrum Master 要参加几个团队的站会怎么办？

4）我不太习惯在其他人面前回答那三个问题怎么办？

4．评审会（Review Meeting）

目的：向团队展示当前阶段的项目成果，会议上应该保证明确地展示了本 Sprint 的业务目标。

活动：

1）简述 Sprint 目标，演示可以实际工作的代码。

2）PO 确认团队增量交付成果是否完成。

小尝试：

1）如何处理 Demo“无法演示”的工作？

2）细节太多，Demo 演示不完怎么办？

5．回顾会（Retrospective Meeting）

目的：帮助团队成员表达自己，总结妨碍团队更高绩效的原因，帮助团队进步。

活动：

1）回顾 Sprint 过程中，人、关系、过程、工具方面做得好的、不好的和需要改进的。

2）寻求解决方案和所需支持，落实行动计划。

小尝试：

1）一边 Break，一边 Retro 可以么？

2）如果总是团队外部的因素影响 Sprint 结果怎么办？

3 套工具

1．Product Backlog 产品功能列表

Product Backlog 就是功能列表，由 PO 维护，需要从业务层面的客户角度进行描述。一个产品 Backlog 的例子：

分类	优先级	Sprint	Point	完成情况	需求名称
支付	3		2		**闪住-支持担保/预付**
		sprint2		完成已上线	story：担保预付
		sprint2		完成已上线	story：埋点
点评	5				**点评列表页接H5 ABT**
		sprint2	1	完成已上线	story：详情页跳列表
		sprint2	1	完成已上线	story：浮层跳点评列表
		sprint2	2	完成已上线	story：H5浮层跳native详情

2．Sprint Backlog 迭代任务

在 Sprint 计划会议上，PO 和团队确定 Sprint 目标以及达成 Sprint 目标需要完成的用户故事，然后团队就这些用户故事进行初步设计和任务分解，得到一个完成这些用户故事的任务清单，这个清单就是我们所说的 Sprint Backlog。

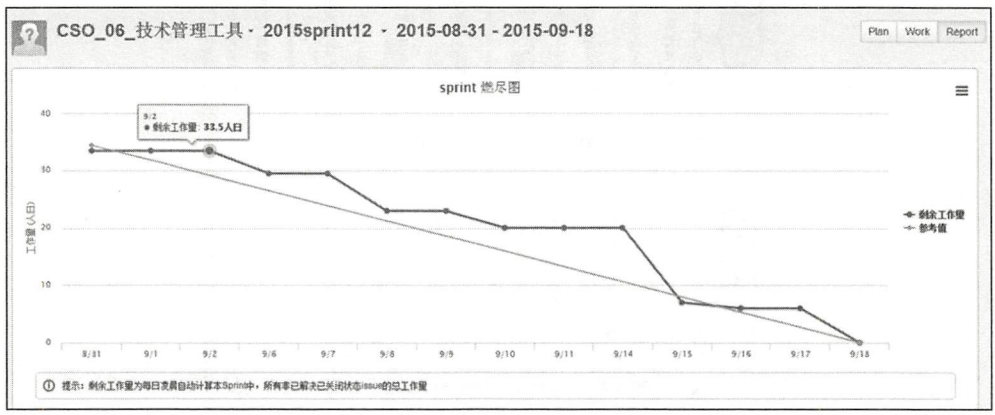
Sprint Backlog 看起来如下图所示：


机票_34_国际主流程Online -
Plan
Work
Report
All
新需求
CR
任务
Bug
Story
创建问题
需求调研池
PRD完成
预排高优先级需求
开发中
历史迭代版本
Idea 82 ISSUE
0M
13
未排序 62 ISSUE
查询页支持过期日期查询
列表页与填写页航站楼信息不一致
机票绑券新产品phase1（外币券）
香港航空A350型号客机商务舱不出售婴儿票
已排序 20 ISSUE
【列表页改版】直飞推荐+直飞日历
【列表页改版】多程超低价
【列表页改版】低价日历
大系统下发二个新的产品的接入：PRC-WS/QNR-WS
如果业务录入超过12条特惠政策 online是怎么挑选展示逻辑的呀？
国籍接口接入（0915）
【列表页改版】推荐——邻近航线、这段时间更便宜、子机场(9.22)
Planning 5 ISSUE
0
26
sprint78
ISSUE
本Sprint中无issue，请拖入issue
sprint78-S1742
0 ISSUE 0 0
Offline-S1713
2 ISSUE 0 26 OK
SOAing-S1626
0 ISSUE 0 0
PRDing-S1625
3 ISSUE 0 0 OK
ISSUE
国际online，模糊搜索改版
unscheduled sp
[story]搜索页展示往返程的临近城市推荐
unscheduled sp
[story]国际提交订单时，如还在订位的界面优化展示
unscheduled sp
ToDo 5 ISSUE
0
20
S171101-S1731
5 ISSUE 0 20
ISSUE
国际验舱发起ssi试订位
unscheduled 2
中转多票新增保障出行10.20
unscheduled 3
【列表页改版】航班卡片
unscheduled 13
online填写页增加航站楼hover
unscheduled 1
机型图增加开关配置
unscheduled 1
Doing 293 ISSUE
6W 3D 1H
1350
S171025-S1742
7 ISSUE 0 11
2017-10-16 ~ 2017-10-20
ISSUE
中转城市名相同特殊处理
unscheduled 5
1
[online]poi新增车站及车站的首页回显
unscheduled 5
机票大促标签展示优化
unscheduled sp
【风控传值】浏览器设备号vid传给下单
unscheduled 1
国际Online三选X 旅行家改版项目phase2
unscheduled sp
多票-填写页重定向判空修复
unscheduled sp
Done 667 ISSUE
26W 7H 50M
S170523-S1720
22 ISSUE 1W 1D 6H
2017-05-15 ~ 2017-05-19
S170516-S1719
29 ISSUE 1D
2017-05-08 ~ 2017-05-12
Offline-0509-S1719
4 ISSUE 0
2017-04-19 ~ 2017-05-09
S170509-S1718
23 ISSUE 1D 2H
2017-05-02 ~ 2017-05-05
S170504-S1717
24 ISSUE 1H
2017-04-24 ~ 2017-04-28
S170425-S1716
23 ISSUE 4D 1H
2017-04-17 ~ 2017-04-21
S170418-S1715
20 ISSUE 0
2017-04-10 ~ 2017-04-14
S170411-S1714
16 ISSUE 4D
2017-04-01 ~ 2017-04-07
Offline-0406-S1714

3．Burn-down Chart 燃尽图

燃尽图能形象地展示当前迭代中的剩余工作量和剩余工作时间的变化趋势，是反应项目进展的一个指示器。一般在每日站会后团队会根据任务的完成情况对其进行更新。

燃尽图有一个 *Y* 轴（剩余工作量）和 *X* 轴（时间）。理想情况下，它应该是一个向下的曲线，随着日期的推进和剩余工作的完成而“燃尽”至零。但是现实情况下，因为各种原因燃尽图往往会出现低谷或者高峰。燃尽图通常如下图所示。



1.1.4 量体裁衣 Kanban vs Scrum

在传统开发模式下，通过建立不同的项目来完成一个需求的完整开发周期，每个需求需要经历 5 个阶段才能上线：评审、排期、开发、测试、上线。在这种模式下，产品经理和开发人员都遇到了什么问题呢？

1．项目间开发时间冲突，优先级不断调整

除了小的需求走 CR，其他需求都是通过立项来管理的，每个需求就是一个项目，在项目经理的管理列表上，同时需要跟进的项目多达十几个，每个项目的生命周期有立项→规划→执行→编码→执行-测试-上线→试运行→结项，共 7 个阶段。

有新需求过来之后，项目经理、产品经理、开发经理和测试经理要一起评估工作量，进行排期，同时兼顾其他需求，调整优先级。如果新需求比较紧急，资源冲突，就要暂停正在进行开发的需求，插入新需求。这样会导致需求的优先级要不断调整，

开发和测试人员比较被动。

2．有些项目开发周期过长，没有短时间内的目标

有些外部对接的需求，从最初需求确定、开发对接以及测试联调，会持续 3 个月甚至更久的时间，中间过程的需求变更、人员变化等都会间接地延长开发时间。在这种持久战中，没有树立短期的目标，团队成员会主动或者被动地等待对方的进度来推进工作，时间久了，心理上容易懈怠。

3．开发进度不透明

每个项目的进度，由项目经理每周召集产品经理、开发经理和测试经理一起开周例会，同步进展的情况以及任务，其他团队成员无法同步到所有的信息，导致信息不能透明传递，容易出现沟通上的误解和延迟。

4．团队比较被动

开发需求由各个开发经理一起开会确定，以及确定排期，然后传达到下面的团队成员。因此，团队成员比较被动地接受这些需求和信息，无法更加直接地沟通。

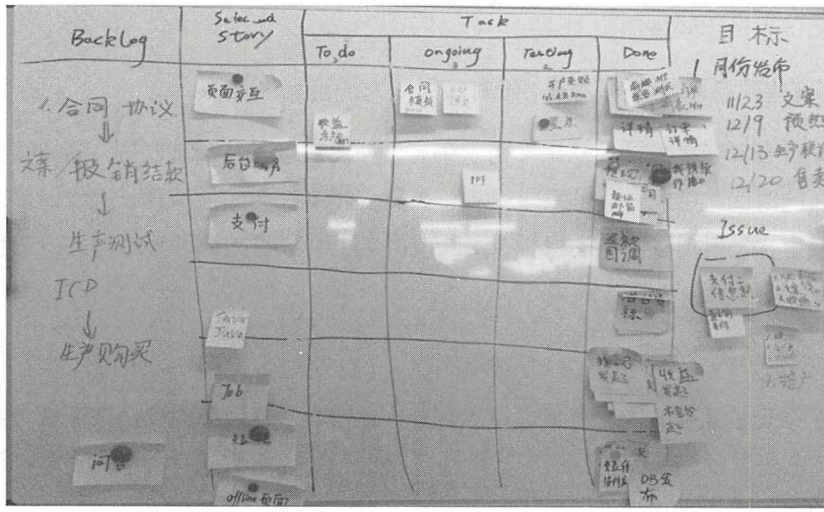
5．人员变动造成进度瓶颈

由于各个职能界限划定比较独立，每个人负责固定的模块，当人员请假或者出现资源冲突时，只能被动地等待资源可用时才能获得进展，这样就很容易造成进度上的瓶颈，进而影响整个需求的交付。针对这些问题，团队选择了向敏捷开发转型，在众多的实践中，选择了更加灵活的 Kanban 模式，迈出了敏捷的第一步。

首先，把所有需要做的需求可视化在白板上。通过卡片把所有的需求按照不同的阶段分别贴在白板上，产生了新的需求持续就加到 Backlog 里。

其次，每天的站会，所有团队成员一起共享进度。打破了之前每周只有经理开会的形式，让团队每个成员都参与到会议中，以保证信息和问题的有效传递。

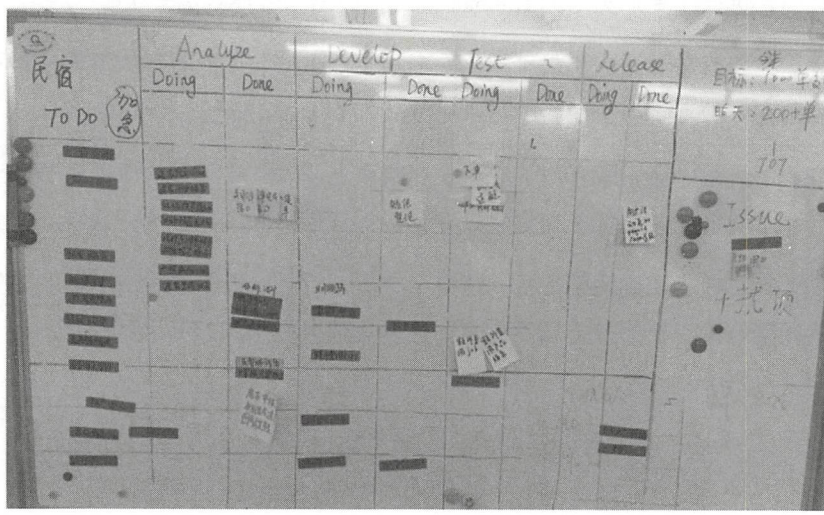
最后，在白板上明确短期目标，让每个成员都看到。根据进展情况以及每天的问题，及时调整对策。在部分人员不在时，鼓励其他人员承担起开发的任务。



经过一段时间的 Kanban 尝试之后，团队慢慢适应了从瀑布到敏捷的开发模式，自觉参加站会，及时跟进需求。但是，慢慢地也暴露了一些问题。

由于需求比较多，而且团队人员比较多，最多时达到 15 人，这样就会导致 WIP 被频繁突破。每次有新需求提出时，仍然需要评估需求池中的各个项目，频繁改变需求的优先级。对于某些开发周期比较长的需求，同样会让团队的目标感不强。这个时候，团队开始寻求比 Kanban 更加适用于团队的 Scrum 模式。

相比较而言，Scrum 比 Kanban 更加规范，Scrum 固定了迭代周期，树立了每个 Sprint 的目标，可以有效解决团队目标感不强的问题。



在这个转型过程中，用到了 Kanban 和 Scrum 两种模式，使得资源分配更加合理，有效地解决了团队遇到的问题，团队的交付能力也大大提升了。

1.2　敏捷能为我们带来什么

产品总不能令客户满意，有没有？

需求那么多，那么复杂，还不停地变来变去，有没有？

为赶进度发布版本，经常加班加点，一点也不开心！有没有？

相信大家都深有体会了，那么敏捷能否解决这些问题呢？

答案是：敏捷不是银弹，不能指望它能解决所有问题。

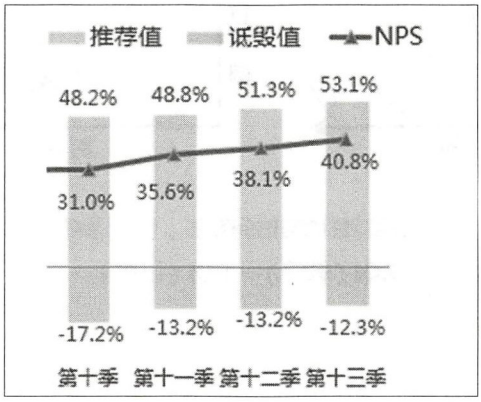
那么，敏捷到底能为我们带来什么呢？

最初我认为，敏捷就是快！快速地交付发布，快速响应市场的变化。经过多年实践我才意识到，敏捷不仅仅是快，更重要的是反馈快，其核心是价值最大化。

敏捷转型围绕以下 7 大价值点，运用独家秘籍招式进行持续改进，团队将会变得更高，更快，更强！

Top 1：更高的客户认可度

每个季度，通过 NPS 倾听用户真实反馈，分析数据，发现产品需要优化的方向，以此来进一步提升客户的认可度。NPS 监测数据如下图所示。



敏捷转型后，当前 NPS 针对上一季持续提升（见上图）。团队把 NPS 数据反馈作为头等大事，对于增长性因素继续发扬光大，对于降低性因素及时与客户沟通，快速交付有价值的产品，减少产品因素对客户造成的影响。

招式：

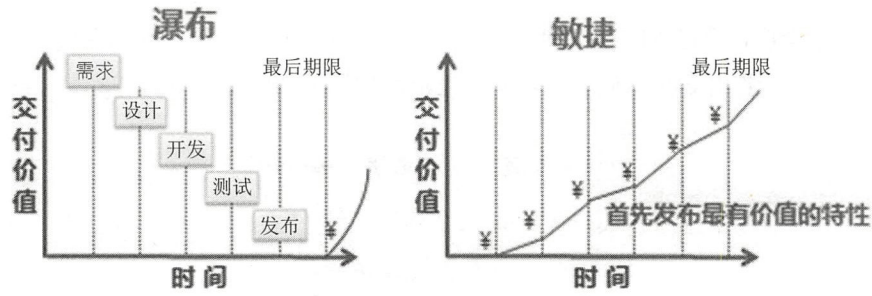
- 达成共识的计划
- 持续交付
- 倾听反馈
- 有价值的交付物
- 及时沟通
- 快速试错

备注：具体招式详见第 6 章“用敏捷思维做大产品”。

Top 2：更高的企业收益

价值驱动，从高优先级的需求出发，尽早开始，尽早交付，并把可工作的产品展示给用户，以获得反馈，验证我们对市场和需求的认知。

铭记：我们手上的资源是有限的，也没什么是完美的，在有限的人生和资源面前，把握住对你最重要的东西，比拥有一切更重要，明白这个道理的时候，你的人生才真的明白了。



产品特性随着时间变长，价值递减

ROI（敏捷）> ROI（瀑布）

招式：

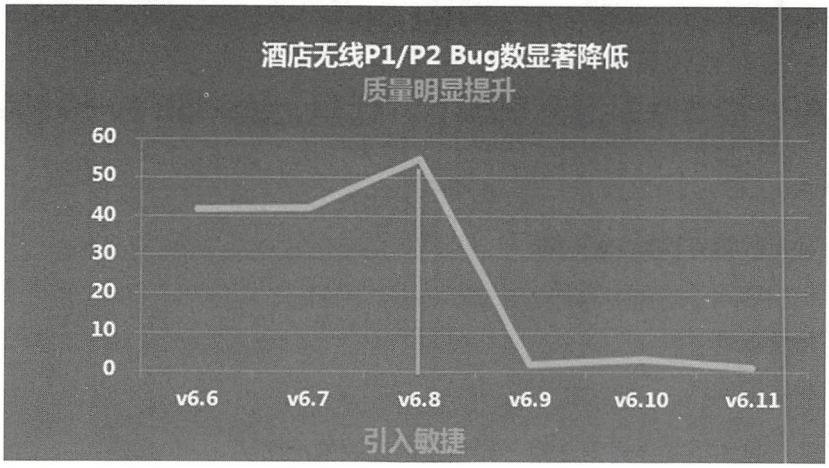
- 市场导向
- 专职 PO
- Sprint 级 ROI
- 优先高价值 User Story

备注：具体招式详见第 4 章“极致敏捷的小团队”。

Top 3：更高的质量

敏捷对质量的改进尤为显著，每个 User Story 都有 DoR 和 DoD，DoR 让技术人员满意，DoD 让产品人员满意，这样才算真正完成任务，不要让没有达到质量要求的任务流到下一个环节。

在酒店无线团队刚刚引入敏捷时，SIT 阶段（上线前）统计 P1/P2 的 Bug 个数高达 55 个，在之前也有将近 40 个，经过两个敏捷迭代之后，P1/P2 的 Bug 数迅速降低到 2 个，并持续维持在个位数水平。



招式：

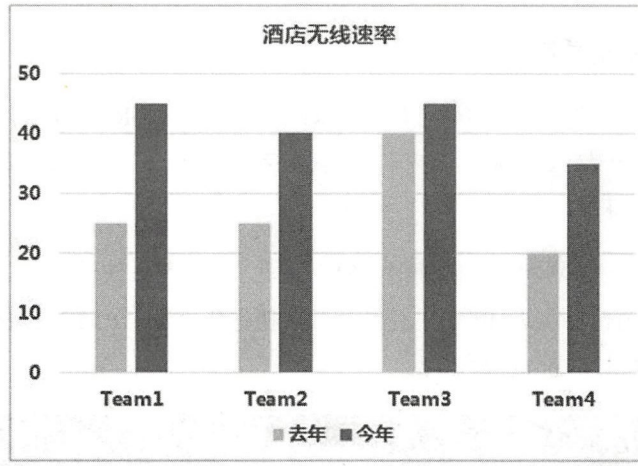
- 定义 DoD
- 单元测试
- 自动化测试
- 持续集成 CI/持续交付 CD

- 提高 User Story 质量

备注：具体招式详见第 5 章“效率为王，唯快不破”。

Top 4：更高的团队生产力

来自 Agile Impact 的报告显示，转型敏捷团队的生产率平均提升 25%，在酒店无线转型的两年期间，团队稳定，基本无离职，团队持续回顾改进，生产力提升 60%。



APP整体速率提升60%↑　代码上线率50%→93%↑

招式：

- 项目数据可视化/Kanban
- 持续回顾
- 持续改进
- 稳定的团队

备注：具体招式详见第 2 章“如何组建高效的小团队”。

Top 5：更快的市场投放

使用商业价值驱动、聚焦产品核心价值，将最重要的目标放在顶层，分解、交付，渐进明细，找出方向，持续优化市场投放策略。

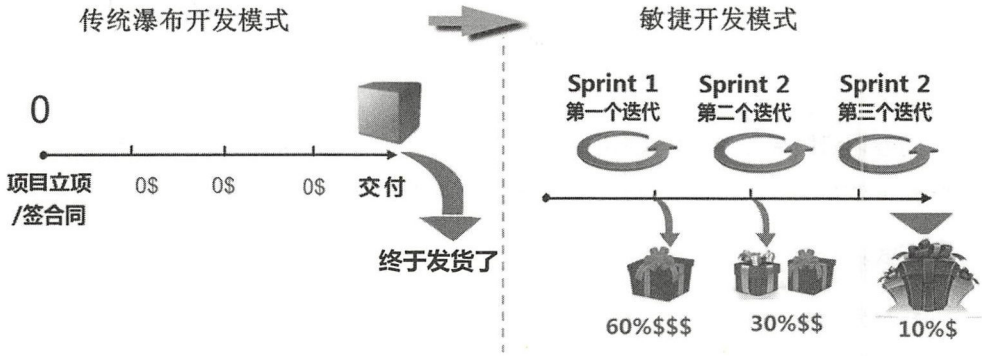
招式：

- 快速响应市场反馈
- 持续市场反馈
- 快速调整计划
- 快速调整优先级

备注：具体招式详见第 6 章“用敏捷思维做大产品”。

Top 6：更快的产品上线

在传统瀑布开发模式下，产品上线至少需要 1 个月，虽然敏捷开发模式下产品研发周期没有变，但每个迭代周期在 1~4 周内，均可产生潜在可上线增量，可以在更短的交付周期内上线对用户更有价值的版本。



招式：

- 固定时间盒（1~2 周迭代周期）
- 增量式交付
- 达成共识计划

备注：具体招式详见第 2 章“如何组建高效的小团队”。

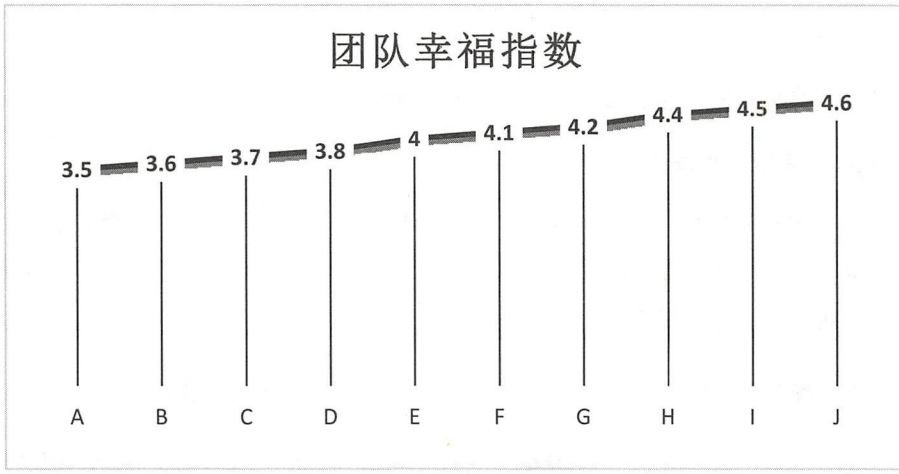
Top 7：更强的团队幸福感

我们的使命是让旅行更幸福。旅行带来的是探索、协作、交流和创造，这些都是人类最高境界的幸福追求。我们追求的是在开放、友爱以及协作的公司文化中工作，这让我们感到快乐。

那么，你的工作真的幸福吗？我们在公司内的研发团队做了如下问卷调研。

1．在自己的团队中，你有多快乐？满分 5 分，您愿意打几分？

2．就公司而言，你觉得有多快乐？满分 5 分，您愿意打几分？



问卷调研结果显示，以 OKR 为目标的内部创业小团队的幸福指数明显高于持续加班的支撑性工作团队，而且开放、有爱的小团队普遍高于 4 分。

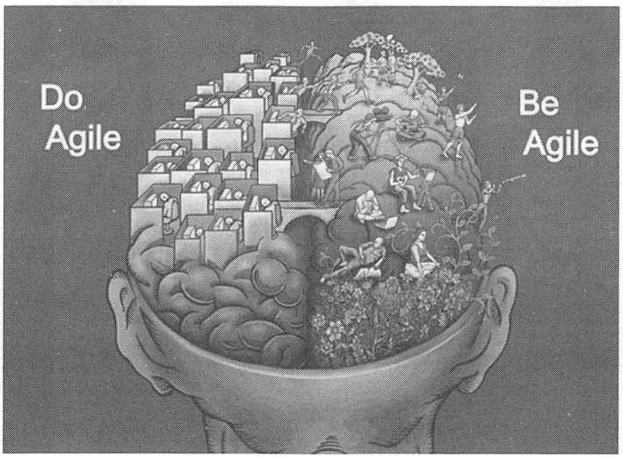
招式：

- 聚焦目标而非人
- 拥抱变化
- 增强成就感
- 增强凝聚力

备注：具体招式详见第 3 章“打造幸福的小团队”。

这 7 大价值点只是一个附属品，不是我们的目标。但这会让我们的团队更灵活、更轻盈、更健壮。

1.3 Be Agile，Not Do Agile



Do Agile 就像我们的左脑，从问题出发，寻找解决方案，实现对产品或者服务的改进。

Be Agile 就像我们的右脑，从客户角度出发，探索发现客户隐藏需求，快速反应，设计体验、产品、服务。

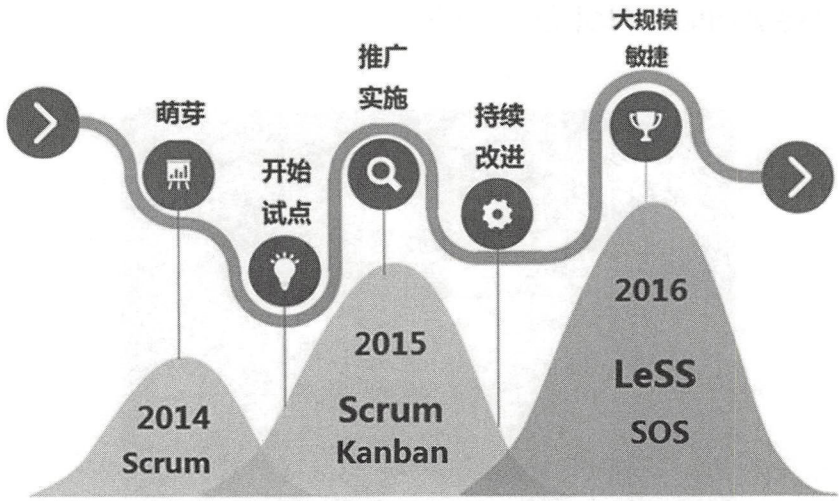
敏捷其实就是提倡我们改变固有的左脑思维，让更多人习惯用右脑思维，变得更敏捷。

1.3.1 改变是我们与生俱来的能力

改变，是我们本能的全面唤醒和回归。

改变，是我们革新和造就自我的最终途径……

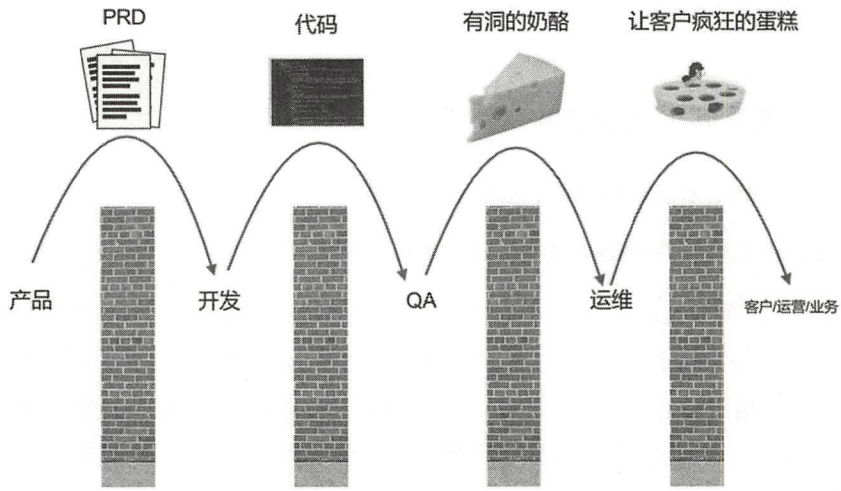
2014 年，各大互联网巨头抢占移动互联网入口，为了应对市场的快速变化，我们怀着不安的心情踏上了敏捷之旅（敏捷转型路线图如下图所示），我们要打破部门之间的界限，组建动态、扁平化的团队，让 PO 来驱动项目，让研发更贴近业务，与业务形成合力，一起达成公司的业绩目标，赢得市场竞争。下面，让我们共同见证改变的力量。



起步——新的旅程

曾几何时，我们想改变世界，却发现周围的一切坚硬如铁。

亦如我们每天乐此不疲地玩着隔墙抛物的游戏，如下图所示。我们的产品经理写好 PRD，写得非常全面仔细，然后抛给我们的开发人员写代码，再抛给 QA 测试，为了赶工期带着 Bug 抛给了运维人员，运维人员拿去上线，最后只会让客户抓狂。



这一次，我们尝试着让产品人员、开发人员、QA 在一起研究什么是我们最薄弱、

最要改善的地方，然后寻求高层领导的支持，探索能否按照垂直产品划分，聚合技术团队，降低开发耦合，提升研发效率。让产品人员、开发人员、QA 坐在一起，仅此一个小小的改变，我们发现周围也在随着改变，开发人员主动找产品人员确认需求，QA 也在产品需求提出时开始了测试设计，测试量大的时候，产品人员也来帮忙，其乐融融，一片祥和。

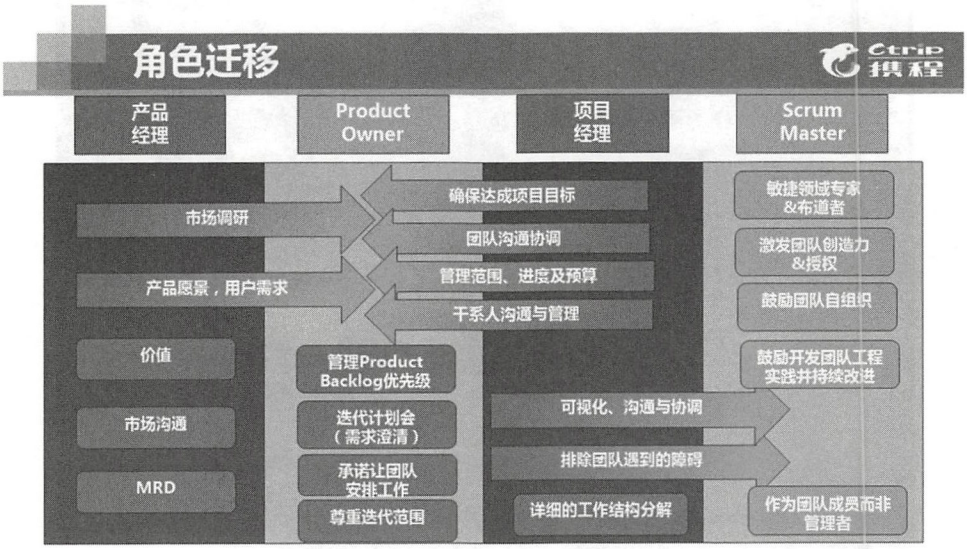
你也愿意打开你的改变之门吗?

尝试——充实的旅程

1．让改变成为一种习惯，从一开始就培养敏捷驱动者

走进 Scrum Team，你会发现，项目经理不见了，去哪里了呢?

原来由项目经理负责的项目范围、进度、预算、沟通协调和干系人管理被赋予 PO，而团队管理的职能转给了 Scrum Master（SM），这对项目经理来说，PO、SM 都是未知的领域，挑战都特别大。那么项目经理如何转变？首先，要由指挥者变成服务者，帮助团队发现研发过程中的瓶颈，并及时采取措施清除障碍。



如果有足够的耐心说服和影响他人，在遇到困难的时候寻求管理层的帮助，在受到外界挑战和动摇的时候，对自己、团队、组织有坚定的信心，能把自己融入团队里，持续改进，那么，你就是我们要寻找的敏捷驱动者。

找到主动转型的敏捷爱好者，重点培养成先行者，让其不断推动敏捷知识的传播，布道敏捷。让越来越多的人受到影响，参与改变，世界慢慢会开始为我们改变。

2．小步快跑：和兄弟姐妹们一起感受在微风中敏捷小跑

有时候常常会想，我整天这么忙，天天加班，也看不到终点在哪里。而在管理层看来却是另一番景象：你整天瞎忙，还不出活。如何让每个人忙的事情有价值？谁也不希望自己是那个“猪一样”的队友。

这时候可以试着用 Kanban 把我们做的事情都展示出来，找找价值流动在哪里受阻，怎么最大化产出？还要让研发人员和业务团队一起跑起来，边跑边看路，及时调整方向。

3．让改变变成习惯，让习惯变成文化，稳固我们奔跑的氛围

君子博学而日三省乎己，在敏捷团队中，养成了每天花 15 分钟进行回顾，而且每日站会对所有团队开放，设立敏捷开放日（如下图所示），以便对敏捷感兴趣的小伙伴与 Scrum Master 或 PO 交流学习。

收获——幸福的旅程

1．团队的变化

打破了职能壁垒，形成高度可视化跨职能的自组织团队，转型 Scrum 团队（如下图所示）。

- 目标一致，规则透明

拆掉了职能部门墙，大家为同一个目标走到了一起，每个人都了解团队正在做的事情，团队成员互为 Backup。每天都有看板和燃尽图来表示项目进展到哪里，是否有延期情况，10 秒钟就可以看清进展，一目了然。对于客户、管理者、团队都是透明可见的（业务部门对研发部门最大的顾虑就是研发部门是个“黑匣子”）。

- 团队成员决定做什么、怎么做

管理层决定目标和方向，由团队决定如何达成目标，团队决定分工协作。

- 需求实现“上中下游”团队成员自主推进

每个人都是需求的 Owner，由 Owner 来驱动上中下游的需求实现，而不再由项目经理来连接各个环节，实现开发过程自运转。

- 打造以产品经理为核心的团队（PO 制）

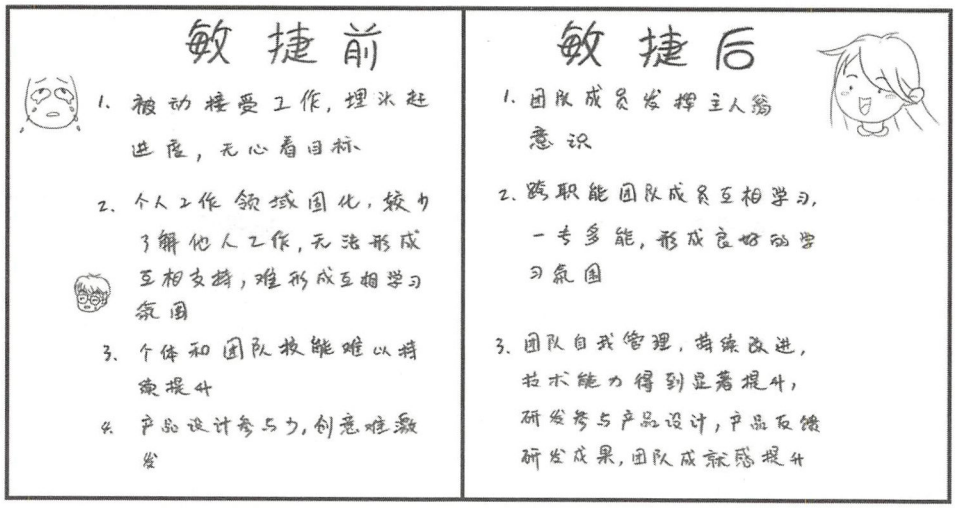
产品的愿景和方向是否正确决定了整个团队的生死存亡。产品经理要对产品“价值”负责，包括用户价值和商业价值。作为优秀的产品经理考虑最多的就是“如何打造用户最喜爱的产品”，而不是让所有人花了那么多精力和时间，却做了一个没有人用的产品。我们需要产品经理产出一个次时代的产品，成为站在风口的发现者，才对得起这帮熬夜加班的兄弟姐妹。

- 追求自动化，提升研发效率，保障产品质量

敏捷团队更加强调不断交付可以工作的产品，为了实现这个目标，要避免依赖手工测试，减少等待和浪费。团队可以通过持续集成、自动化测试不断提升研发效率和保障产品质量。

2．成员的变化


敏捷前
敏捷后
诏书
我就做什么
我认为已经做正确
我们还必做好不好？
I have a good idea
我做完我负责的部分了，你的…
我哪知道
I where know
我做完了，还有什么可以帮其他小伙伴的吗？
我不是一个人在战斗
I am not a person in a fight
fighting



我也想人生快速迭代，怎么办?

我们在努力解放你的思想，但只能为你打开一扇门，你必须自己走出你的“庐山”。有意识地关注自我，愿意积极面对问题，主动适应变化甚至创造、改变，即使这可能意味着在某个时间段中会使自己陷入混乱和困惑，可能会感到恐惧而退缩。然而，这正是你走上正轨的标志。

加入“敏捷总动员”大家庭，尝试使用敏捷经营自己的生活、职业或事业，跟我们一起变得爱学习和有思想，变得懂生活和有目标，变得要行动和有结果，成为一个快乐、高效、平衡的人。

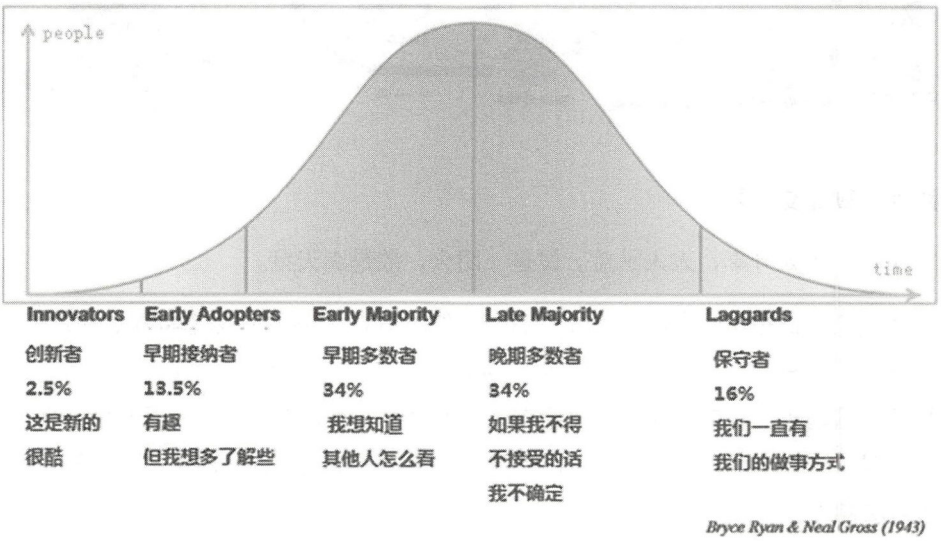
扫一扫加入【敏捷总动员】

1.3.2　敏捷变革的 10 个小套路

作为敏捷变革运动的推动者，不仅要让团队主动拥抱变化，还要提供一个高效有趣的学习氛围帮助团队转型。

自团队引入 Scrum、Kanban 和 XP 等敏捷模式后，经过两年不断尝试和落地实践，成功转型为敏捷团队的占比高达 80%。我们回顾总结了 10 个实用小套路，如果可以帮助你（在你的组织里）引入新的想法，不妨一试，而这仅仅是一个开始！

我们相信，人是可以改变的，鼓励每个人都成为创新者。接受度曲线如下图所示。



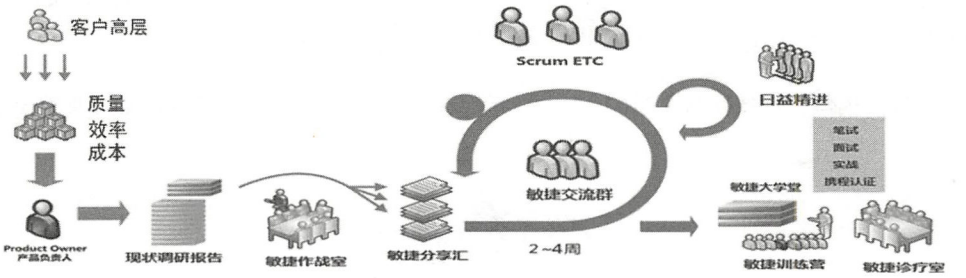
作为敏捷布道者，要影响和改变他人，而这通常是极为困难的，我们无法改变他人的人格和性格，但是我们可以改变人的思想、理念、习惯和行为。

敏捷转型初期，为了帮助刚刚接触敏捷的小伙伴快速学习，我们尝试用 Scrum 模式边学边用，不断在组织中让其生根、发芽、开花、结果。

首先，挖掘具备敏捷的“土壤”，寻求支持者和那 2.5%的创新者，跟他们一起树立标杆团队（最佳敏捷实践奖）。然后，将成功经验分享到敏捷交流群，吸引更多接纳者加入，而且每天都要有新鲜玩法（日益精进），还有专门的 Scrum ETC 组织解决转型过程中的各种问题。

最终，将每两周产出的成果在“敏捷大学堂”宣讲，影响来这里的早期多数者。每个迭代周期结束时在“敏捷诊疗室”进行回顾，思考还有什么可以做得更好，沉淀的成就在“敏捷训练营”培养更多的敏捷布道者，让敏捷无处不在。

敏捷交流学习路线图



套路 1：敏捷交流群

思路：人人分享，人人受益。聚是一团火，散是满天星。

敏捷交流群是一个有趣、有用、有行动的交流群。群内可以求教问题，分享干货，共同研究新方法，让我们的工作更轻松。群中也会定期举行线上和线下互助活动，帮助初学者迅速成长及更好地参与讨论交流群的建设。也会特别邀请业界大牛分享优秀敏捷实践，让进阶者可以持续成长，也可以帮助更多团队开启敏捷意识，培养他们的问题解决能力。

套路 2：敏捷大学堂

思路：接地气，重实践。借它山之石，可以攻玉。

敏捷大学堂，每两周一次，用于跨团队间经验交流分享，帮助解决团队在实际转型过程中碰到的各种问题。主要分享敏捷管理和工程实践经验，包括敏捷转型、CodeReview、自动化测试、持续集成、持续交付等内容。

敏捷大学堂

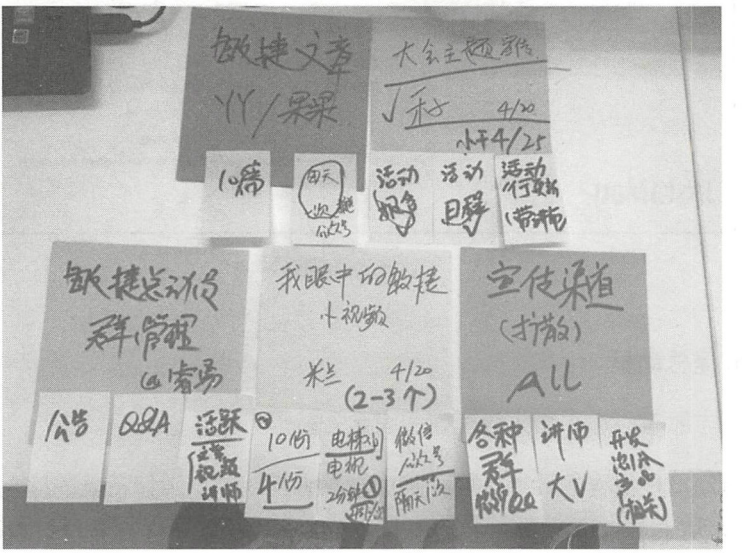
套路 3：敏捷总动员

思路：志愿者心态，赠人玫瑰，手有余香。

敏捷总动员是携程的敏捷之旅，是由志愿者组织的、敏捷粉丝的盛大聚会活动，在每年 5 月 20 日这个特别的日子举行。致力于为广大敏捷爱好者提供高效、有趣的

敏捷开发学习途径，在上海技术圈子内推广敏捷开发思想和实践，帮助企业更好地实施敏捷。

在敏捷总动员组织者里，没有领导。大家为了 5·20 敏捷总动员这个共同的愿景走到一起，大家一起梳理出要做的事情，主动领取，没有人指派，如下图所示。当遇到困难时，会主动寻找资源和帮助，而不是怨天尤人。这就是志愿者心态，主动跳出舒适区，主动给自己找麻烦。



套路 4：有趣的茶话会

思路：Lean Coffee，有趣又有料。

Scrum Master 小伙伴们每月进行一次聚会，主要交流在各自团队落地的成功模式或碰到的棘手问题。每次聚会都有一个特别的主题，由一位主聊准备，陪聊发表观点。

在一次聚会中，有人提出，我们都知道 Scrum 是橄榄球运动中的争球，那到底我们可以从这个运动中借鉴什么到工作中呢？刚好 Scrum Master 团队中有位足球迷，主动要求分享“从足球战术聊聊团队协作那点事儿”，没想到当天还带来他心爱的球衣，穿着球衣聊足球好有代入感。

聚会开展了一年多，才发现我们的茶话会居然跟 Lean Coffee 不谋而合，它是 2009 年在西雅图开始的，Jim Benson 和 Jeremy Lightsmith 想组建一个小组讨论知识工作中的精益技术而使用的模式。

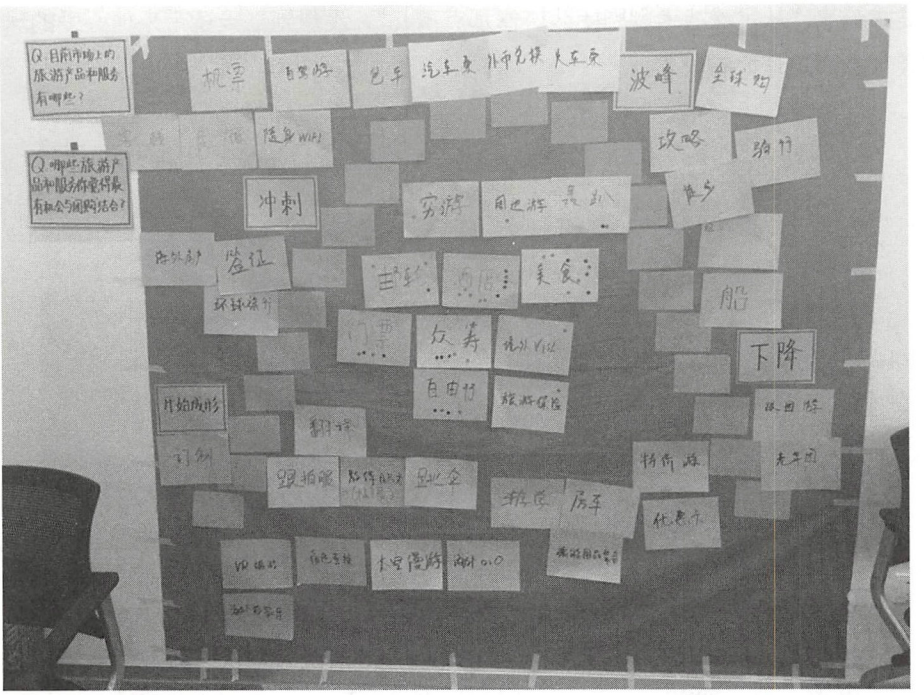
聚会中每人都可以提出想要讨论的话题，并票选出大家最想参与的话题（每人 2 票），按照票数排列优先级并迅速进入讨论阶段，每个主题限时 5~8 分钟，小组成员可以随时表决继续或者转换话题。

不管怎样，我们的茶话会一定要有趣还要有料，这是关键！

套路 5：敏捷创新工作坊

思路：推陈出新，打造用户喜爱的产品。

敏捷创新工作坊是一个鼓励员工公开创新的活动，它打破各部门、各中心、各公司之间的壁垒，互惠共赢，是对现有项目体系的一个补充，让公司项目体系更完善。



套路 6：精益创业沙龙

思路：精益创业，小步快跑、快速迭代。

精益创业沙龙是“内部创业”的讨论咖啡厅，是宣传自己理念的小酒馆。在更小的范围、更短的时间内，实现更自由的模式，这里没有大会议，没有讲台，没有领导，只有讨论和专家，更接地气，更有趣，更轻松，更是一个志同道合的小伙伴们的聚会。精益创业沙龙中常用到的精益画布如下图所示。

套路 7：Scrum ETC

思路：作战司令部，文化宣传阵地。

Scrum ETC 是发起、鼓励与支持公司引入 Scrum 的敏捷社区，指导实施变革并为变革创造活力与激情。它的存在是为了创造一个敏捷文化的氛围，让先行团队的成功带动更多团队的热情加入转型的阵营。

ETC 的成员通常不超过 12 人，公司级 ETC 包括各产品线 CTO、CPO、项目总监等参与 Scrum 转型的最高级别人员。产品级 ETC 包括 PO、SM、技术 Leader 及愿意对团队变革付出的敏捷爱好者。

套路 8：敏捷全程教练服务

思路：扶上马，送一程。

每 3 个月为一个疗程，指导战略重点产品敏捷技术与管理，并在团队内部培养至少 1 名 Scrum Master。

针对团队现状提出研发管理及改进方案，并明确改进目标（包括定性和定量目

标），制定整体改进计划。基于整体改进计划推进实施，3 个月后，检视改进目标达成情况，决定下一步改进方向与计划。

套路 9：敏捷训练营

思路：黄埔军校，SM、PO 大本营。

敏捷训练营是针对 SM、PO 和 Dev Team 的专业性培养及认证。在这里，每个角色都可以收获更大的价值。

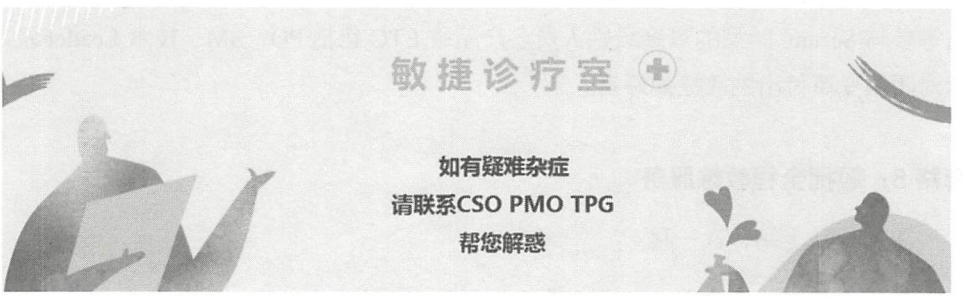
SM：可以了解 Scrum 框架的工作方式，从 SM 的视角亲身体验 Scrum 的工作过程，深入探讨在组织中实施 Scrum 将会面临的方方面面的问题，以便引导自己的团队开展 Scrum 转型。

PO：会发现写 Story 以及维护产品 Backlog 并不是 PO 的主要职责，应该调动团队一切可以调动的力量帮助 PO。同时关注力要面向市场、贴近客户，如何快速通过反馈来找到产的方向，并不断迭代产品。

Dev Team：体验真实敏捷 Scrum 团队中的高效协作方式，理解测试先行的理念，学会测试驱动开发方法，学会识别代码 Bug，并通过小步重构来改善架构与设计，培养编写整洁代码、有效单元测试的习惯，理解持续集成和分层自动化测试。

套路 10：敏捷诊疗室

思路：望闻问切，对症下药。



定期设（内外部）专家坐诊，根据对团队的诊断，含现状分析（基于 360 度访谈、数据分析、现场观察等）、问题诊断、团队成熟度评估，完成与 Scrum Master、PO 及关键干系人的沟通，给出改进和行动计划建议。

彩蛋：我眼中的敏捷

套路也好，规则也罢，关键都在于人！更高效、更快乐的工作，而不只是运用敏捷！扫一扫看小视频“我眼中的敏捷”。

1.4　一步步来，敏捷就不会远

Scrum Master、PO 和团队从零开始接触敏捷，边学边用，一步一步从实践中走出来，蜕变为最优秀的自己，跟着我们一起记录下属于你的心路历程吧。

1.4.1　一个 Scrum Master 的成长札记

酒店无线团队转型两年，从最初的大团队瀑布模型，到现在的小团队 Scrum 模式，这个过程不仅见证了团队的成长，也看到了自己的进步。起初我也只是一名普通的项目经理，在顺应公司各种变化的同时，不得不逼自己去学习，去成长。我想，引用王国维的三重进阶说法，用来描述我的成长札记，似乎比较契合。

第一重境界：（知道自己不懂）昨夜西风凋碧树，独上高楼，望尽天涯路。

第二重境界：（不知道自己懂）衣带渐宽终不悔，为伊消得人憔悴。

第三重境界：（知道自己懂了）众里寻他千百度，蓦然回首，那人却在灯火阑珊处。

第一重境界

昨夜西风凋碧树，独上高楼，望尽天涯路。

2015 年的一天，团队内部的各种矛盾似乎在一时间全都爆发了，需求的变更，开发测试的矛盾等，让我以及各位 Leader 都意识到，团队的问题已经很严重了，我们不得不去做些什么来改变现在的情况，这个时候想到了敏捷！

组建敏捷团队，大家位置调整在一起；以产品线划分组织架构，让团队更专注。计划会议调研好所有技术方案，会议上无论是开发或测试人员，都需要了解每个用户故事的方案。团队所有人一起来承诺 Sprint 范围，尊重每个人的意见……我不知道为

什么这样做，只知道书本上说的要这样做……

第二重境界

衣带渐宽终不悔，为伊消得人憔悴。

运行敏捷两年，一直都是在发现问题和解决问题中循环，老问题刚解决，新问题又冒出来，似乎就没有停歇过。而很多问题的解决办法，是很难在书本中找到答案的，只能一步一步去尝试，找到最适合团队的那种办法！

这两年来，团队确实有明显的改善，积极性提高了。团队内部沟通比以往提升很多，信息透明化，工作也井然有序，任务的安排也可以按照自己的意愿来决定，有自主决定的能力。资源浪费减少，生产力也较之前有明显提升……

这些改变不是一蹴而就的，是这两年大家的共同成果，复盘这两年的点点滴滴，我自己陷入了沉思，团队的这些改变是怎么发生的呢？从来没有弄明白过为什么要使用敏捷，为什么要使用 Scrum，我一直在思考这么几个问题：什么是敏捷？敏捷到底改变了我们什么？敏捷项目管理与传统项目管理有什么区别？如果现在退回到两年前，团队不使用敏捷，一切是否会不一样呢？

做了两年，我似乎从来不知道为什么要这么做……

第三重境界

众里寻他千百度，蓦然回首，那人却在灯火阑珊处。

继续往前探寻，探寻为什么要这么做，现在似乎明白了，万变不离其宗。敏捷宣言，简单的四句话，囊括了我们很多内容。

1．个体和互动高于流程和工具

团队坐在一起，鼓励面对面沟通，多说话，少打字，提高沟通效率。小团队作战，大家一起商量策略，为了共同的目标而努力！

2．工作的软件高于详尽的文档

再详细的文档，也需要花时间去写，如果团队一起商讨出来的方案，每个人都发表自己的看法，真正地参与到其中，那么详细的文档就没有必要。取而代之的是可工

作的软件，让大家都关注在可工作的功能上面，而非单平台的技术方案上面，提升团队整体的全局观！

3．客户合作高于合同谈判

客户不是敌人，而是我们的合作方，大家需要的是信任，而非是条条框框的约束。人人都是产品经理，无论是开发人员、测试人员，还是业务方，大家的目标必须要统一，只有相互信任，才能产出优秀的产品！

4．响应变化高于遵循计划

市场瞬息万变，再完美的计划，也免不了有变更，团队必须有相应变化的能力，而非一味地遵循计划。我们是具有思考能力的人，而不是只会编码的机器，就像打仗一样，我们不知道什么时候敌人会来，那唯一能做的就是时刻准备着应战！

响应变化很重要！作为一个有两年转型经验的 Scrum Master 来说，有几点心得可以跟大家分享。

以身作则：己所不欲勿施于人，自己一定要在工作中起到标杆作用，身体力行告诉大家，自己能做好的事情，你们也可以！

知其然，更要知其所以然：无论是我们所用的知识，还是平时遇到的问题，一定要追根溯源，不要轻易放过任何一个问题出现的过程，弄明白一个问题远比你从书本上学习几个知识更有用。

该放手时就放手：不要让自己变成一个保姆，事无巨细地去管理团队大大小小的事情，要相信团队的能力，十几个人的力量绝对比一个 Scrum Master 的力量要大，该放手时就放手，相信团队！

尊重与信任：团队里面任何人的意见和看法都是弥足珍贵的，有正面也有反面的，正面的声音是对我们的认可，反面的声音更能促进我们成长，要尊重每一个人！要相信大家的本意是好的！每个矛盾后面都有一个原因！

面对问题，不要逃避：团队内部的问题层出不穷，不要害怕，不要担心，要勇敢地去面对，越是难啃的骨头，越不能退缩。敏捷对于团队来说是一次蜕变，对个人来说，更是一种突破！

1.4.2 如何成为一名优秀的 Product Owner

加入携程酒店前端产品团队一年有余，我深切地感受到 Scrum 的奇妙魅力，并有幸以 PO 的角色，主导着产品方向，努力为用户提供最优的体验，让团队价值最大化。本节将结合我的 Scrum 实战经验，分享我对 Product Owner 的理解和思考。

OKR 指导敏捷开发

产品的使命和任务，必须转化为目标。那么，如何管理好产品目标，是整个产品生命周期和敏捷开发过程的关键。

1．合理制订目标

对于像携程酒店频道这样一个较为成熟的产品，制定目标时则要充分发挥全局思维，找到有潜力且影响面足够大的着力点。可以从是否符合当前公司战略，是否与业务匹配度高，是否有事实数据支撑等多个方面，衡量目标是否合理。

2．关键结果可量化

管理学大师告诉我们，“如果你无法衡量它，你就无法管理它”。完备的 OKR 不仅要制订合理的大“目标”，也要确立可衡量的“关键成果”，即 KR。像“提高产品质量”这种就是不合格的，像“酒店列表页到酒店详情页流程转化率提升 5%”这种才是合格的。

3．定期复盘

OKR 要定期复盘，内容主要包括目标是什么，这个目标是如何确立的，当前进展是什么，遇到了什么问题，如何解决的，接下来的进度是什么等。定期复盘，也会帮助我们及时发现 OKR 的问题，果断做出调整。

关于需求挖掘

1．从业务出发

作为业务型产品经理，需要对行业了如指掌，这将保证我们能够全面地思考和设计产品。作为酒店频道的产品经理，我需要了解整个酒店行业。包括：酒店行业当前发展阶段和规模、酒店集团品牌的组成和覆盖、酒店的基本物理结构、酒店与 OTA

的合作模式等。熟知这些信息，帮助我快速地融入了业务，为日常工作沟通、掌握业务模式和诉求、规划符合现实的产品方向等都提供了坚实的基础。

2．从用户和场景出发

什么样的人会住酒店？答：商务人士、情侣、亲子等。

什么样的场景会住酒店？答：出差、约会、旅游等。

商务人士有哪些特点？答：中青年为主、男性为主、可接受的价格范围相对明确等。

出差场景有哪些特点？答：机场车站接送、开发票、传真和会议室等。

如何结合用户画像和场景特征挖掘需求，通过以上四问四答，一目了然。

从数据出发

数据驱动，这四个字老生常谈。比如分析用户浏览各页面的点击数据，可以知道用户在购买行为中关注的信息和关注程度，便可以指导设计页面的结构、信息呈现侧重点。比如分析用户下单时间、价格区间等订单特征，可以知道不同用户的购买时机和诉求，便可以指导从时间、空间、基础信息等维度设计功能划分、个性化推荐等。

关于需求优先级的决策：

1．需求要贴合目标

需求优先级的把控质量，决定着团队和产品发展的效率。需求提出后，要反复审视需求是否属于目标计划的一部分，是否有助于完成当前的 OKR。需求挖掘阶段，保证需求的质量；需求优先级决策阶段，则是对需求再次去伪存真的过程。

2．需求要符合公司整体战略

作为 Product Owner，除了管理好自身的原发需求，还要正确合理地处理来自各业务方的需求。判断各方需求的优先级，需要从战略角度衡量。比如处于高速成长期的产品，拉新需求优先级要比界面优化需求优先级高。

3．ROI=收益-成本

确定需求优先级时，一般以收益指标降序排列，往往很容易忽视每个需求的开发

成本。将资源聚焦在高收益低成本的需求上，从而实现收益最大化。

小结

成为一名优秀的 Product Owner，任重而道远。借此文章，祝愿所有奋战在互联网前线的小伙伴们，都能找到高效科学的产品管理方法，实现产品和自我价值。

1.4.3 缔造最佳敏捷实践团队

在紧密团结的敏捷团队里工作是让人兴奋的，但这样的团队也不会平白无故冒出来，团队需要时间才能凝聚起来。那么，如何让团队自觉、自律，变得高效，成为卓越的团队呢？

敏捷团队发展工作坊

我们会定期召集公司 Scrum Master 小伙伴来敏捷团队发展工作坊坐坐。在这里，不仅可以分享成功的敏捷落地经验，还可以群策群力一起探索敏捷转型过程中碰到的最棘手问题的解决方法。通过工作坊的方式希望 Scrum Master 帮助团队找到最舒服、最高效的工作方法，让团队更健壮。

澄清现状

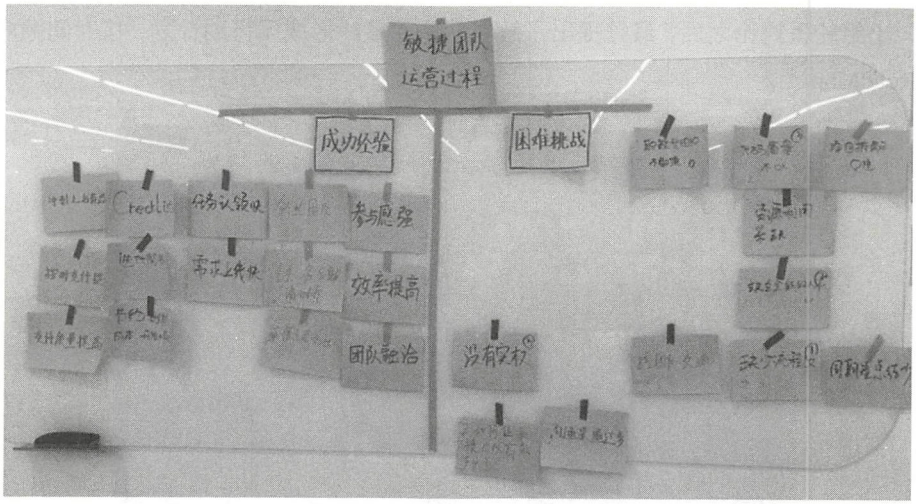
问题：在敏捷团队转型过程中的成功经验和困难挑战有哪些？

1．ME：个人思考并自己记下来。

2．WE：3 人一组讨论分享，并将大家达成共识的内容记下来。

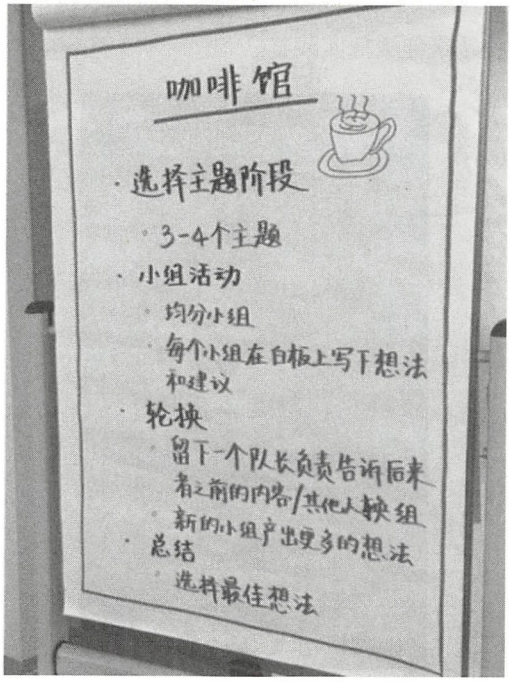
3．US：3 人小组派代表，面对全员展示自己小组的内容。

4．最后，小组或个人投票，选出 4 个关键困难挑战。

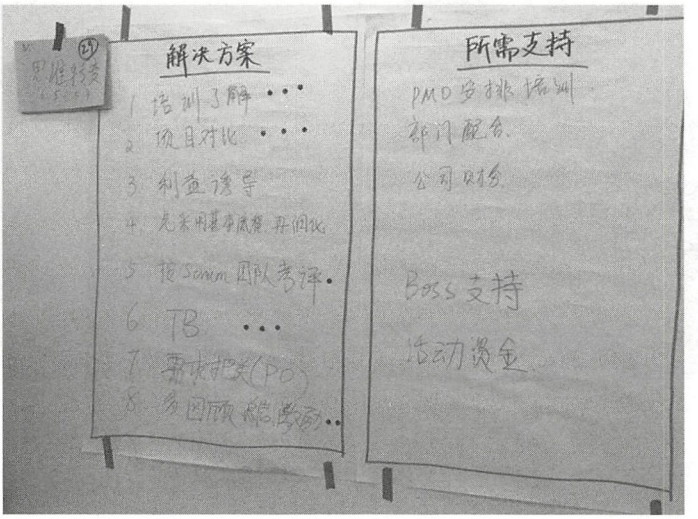


解决方案

“敏捷思维转变”的投票高居榜首，这确实是一个难题，每个团队的情况都不一样，碰到的问题也多种多样。那么，如何解决呢？这里，推荐大家使用咖啡馆的方式引导团队一起寻找问题的解决方法和所需的支持。

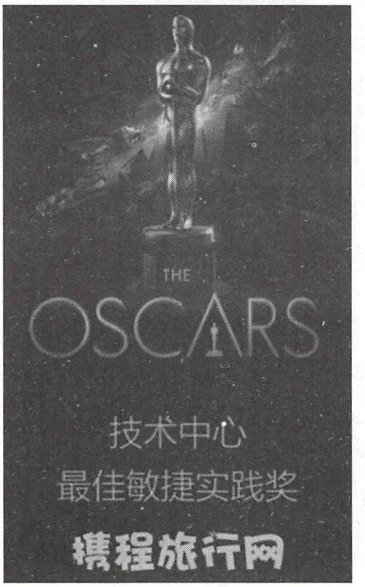


小组轮番讨论之后，最终提出了解决方法并最终形成了行动计划，按时间顺序贴在 RoadMap 上。



设立最佳敏捷实践奖

为了表彰敏捷团队取得的突出成绩，鼓励其不断追求技术卓越，为公司创造更大的价值，我们特别增设了最佳敏捷实践奖。



敏捷成就等级（附 1：敏捷成熟度）高于 3 星的团队（附 2：星级团队）就可以自由申报，入围的团队首轮要经过大众评审，然后才能进入答辩环节，最终 PK 成功的团队将获得 MVP 称号并获取“奥斯卡小金人”。

大众评审：手机在线投票，可微信转发。这不只是在刷存在感，更多的是团队优秀实践和个人事迹的传播，看看 5000 多人次的票选结果，会影响多少团队加入转变的行列。

ETC 评分：答辩环节比较特别，所有参选团队及个人以 Pecha Kucha（20 页 PPT，20 秒/页）的形式进行成果及收益展示，接受 ETC 提问，ETC 在评选标准范围内评分。

拿着“小金人”站在年会上的那份荣誉感，当在朋友圈炫耀团队成就的那一刻，已然拿到特权，坚持做更有成就感的事，也激发了心灵成长带来的那份仪式感。

正是有了这份仪式，我们的生活才不止眼前的那些苟且，人生才有了节点，有了值得回味纪念的“小确幸”，才能在白发苍苍回忆一生的时候骄傲地说，这辈子也曾生动过，存在过。

附 1：敏捷成熟度

敏捷成熟度，是为正在推进敏捷转型的小伙伴们提供参考，评估团队是否是敏捷团队，以及有多敏捷。根据评估情况了解团队的强项、弱项并进行提升改进，激励其向高度敏捷化团队发展。

敏捷成熟度为度量开发团队敏捷程度定义了一套指标，它涉及团队结构、测试自动化、过程组织、信息共享和绩效激励等几个方面。每个指标有一个分值，合计 100 分。通过对团队某一段时间里的表现应用这些指标，我们能得出 0~100 分之间的一个分数，此分数即该团队的敏捷成熟度得分。

附 2：星级团队

为鼓励大家积极向高度敏捷化团队发展，我们根据敏捷成熟度得分把团队分成了 6 个星级。

不低于	星　级
0	★
10	★★
25	★★★
45	★★★★
65	★★★★★
85	★★★★★★

NO.2
团队篇

将亲身实践作为出发点，结合自身痛点共同探讨：如何组建高效的小团队，打造幸福的小团队以及成为极致敏捷团队的方法。本篇将让你更加清晰地认识和探索充实的敏捷之旅，持续收获幸福和成长。

第 2 章

如何组建高效的小团队

哈佛心理学家 J. Richard Hackman 曾表示：“大型团队一般不靠谱，最终只是浪费个人时间罢了。”

Hackman 最重要的研究并非与团队人数相关，而是着眼于当团队成员增加时，人们彼此间千丝万缕的联系。每当有新成员加入，团队的整体协调成本就会增加。比如，团队是 8 个人，其沟通渠道是 8×（8–1）/2=28，假设增加了一个人，变成 9 个人，其沟通渠道就变成了 9×（9–1）/2=36，虽然只增加一个人，但是沟通渠道却增加了 8，可想而知成本也大大增加，当团队每增加一人，沟通渠道是呈非线性几何形增长的。所以管理就是要解决成员之间的联系。

2.1 小团队，公司组织的未来

“微信之父”张小龙在微信内部的“领导力大会”上表示：大团队大平庸，小团队高效率，在开发高创造力的产品时，按部就班会让一个非常平庸的团队在不知不觉中用了一些非常平庸的方法做出来一个非常平庸的产品，唯有“敏捷项目”管理才能出奇制胜。

携程创始人梁建章带领携程二次创业，当时公司规模庞大，凡事都要走到决策层，是一个非常缓慢的过程，特别是一些新业务，必须跟着市场走，所以设立了 SBU 这

样的 CEO 负责制的事业部，决策效率会大大提升，资源也会更到位，产品、技术、服务各方面结合得更紧密。在公司层面我们是个很大的团队，但是内部又是由一个个有活力、被充分授权的小团队在运转。

可见，小团队会是未来公司组织变革的趋势。

随着公司规模的扩大，“人”的因素必然将越来越复杂，到了一定程度的时候，就必然会产生内耗，各种各样的内部矛盾，即使再厉害的老板，也只能缓解这个过程，无法解决这个问题。如何缓解呢？那就是流程和规章制度，所以我们会看到各种各样的“规则”层出不穷，各个团队风格迥异，各有特色。某种程度上来说，流程和规章制度并没有什么问题，但是对于互联网公司来说，节奏越来越快，软件越来越复杂，很大程度上需要发挥人的创造力，冗杂的流程和规章制度，只会让人束手束脚，无法发挥出最大潜能，尤其是在面对重大决策时，只能层层审批，拖慢进度，很有可能就错过了黄金时期。

在携程内部有个创新工场，专门用来孵化新的项目，项目组采用的是垂直管理的方式，整个项目组里面有研发、测试、产品、运维等角色，能够独立运行一条产品线，项目组的所有人为这个产品的成败负责，小团队人少效率高！酒店、机票这种大 BU，团队太大，内部耦合比较严重，但是为了顺应公司的发展，将组织架构进行整合，去除中间层，更加扁平化管理，职能汇报关系不变，将团队拆分成一个一个的 Scrum Team，按照产品线来划分，不同角色的人工作在一起，共同为了产品的目标而努力！

无论项目组制，还是产品线制，都是为了将组织扁平化，减少过程中繁杂的审批过程，小团队作战，激活组织的活力！

2.2　组建小团队，你可以这么玩

酒店、机票作为携程最早的 BU 之一，业务已经相对稳定，如何让团队始终充满激情，持续创新，快速响应市场的变化呢？

2015 年年初，酒店和机票无线团队开始尝试敏捷转型，将大团队拆分成小团队，采用迭代、小步快跑的方式进行敏捷开发，旨在提高开发效率和响应能力，以适应不断变化的业务需要。麻雀虽小五脏俱全，各团队兵马粮草配备齐全，不论何种需求都能积极应对。

经历了两年的敏捷转型尝试，我们将 APP 重新进行了梳理，划分成了不同的产品线，每条产品线都对应唯一的 Scrum 团队，团队内所有人对自己的产品负责。团队与团队之间相互竞争，营造“内部创业氛围”，为达成目标再一次吹响冲锋号。

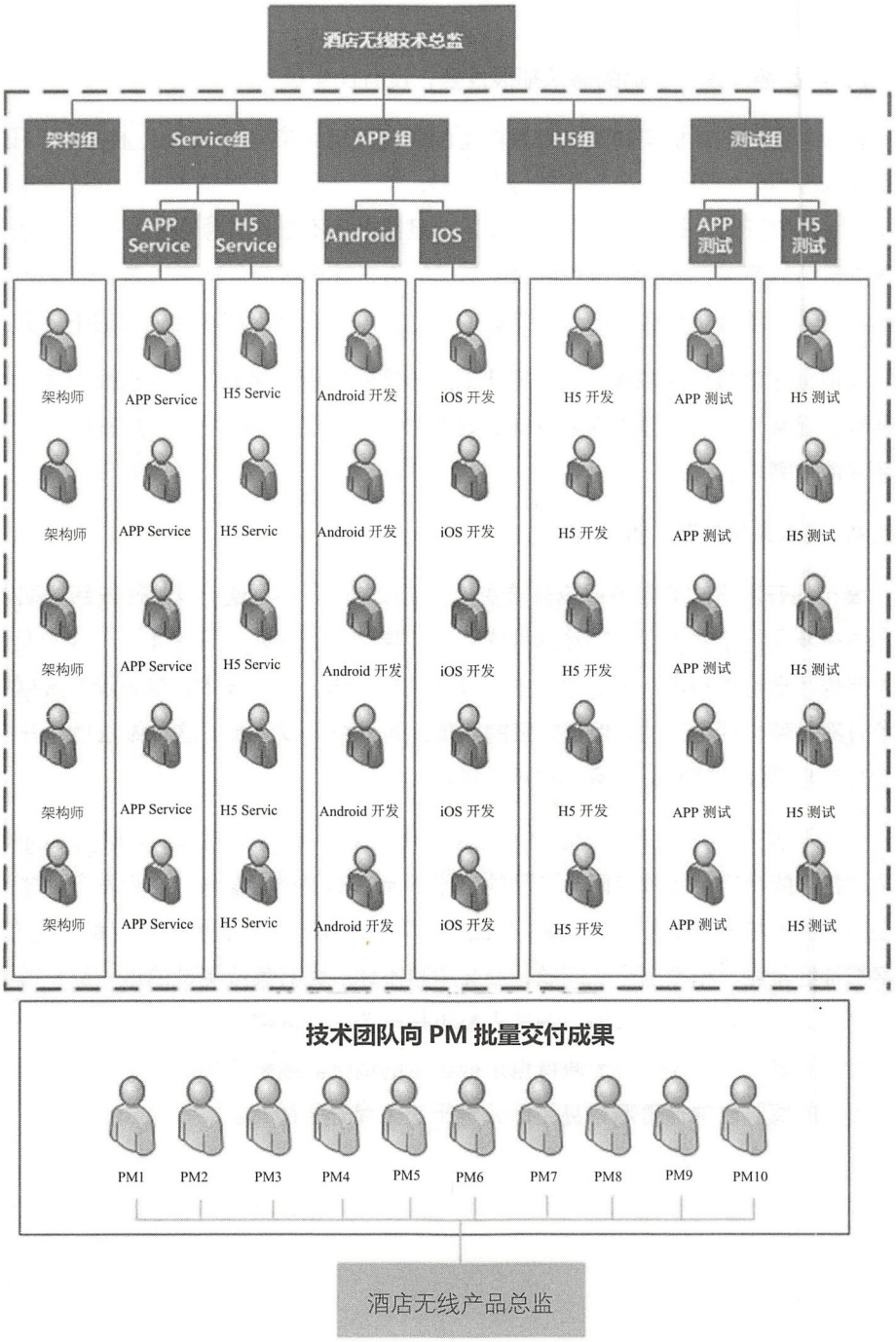
下面就来分享我们打造高效团队的心路历程！

为什么要做转型

1．团队大，沟通起来心好累

当时的酒店无线团队，APP 开发 60 人左右，1 个月 1 个版本，产品经理给出 PRD，60 人一起参加需求评审会，经过需求评审会、需求 PK 会、范围确认会等才能最终敲定版本 APP 的需求列表。开发过程中还需要进行架构方案评审会议、测试用例评审会议等，遇到产品需求变更，以及新增需求时，还需要进行 Change 评审会议、Change 架构评审会议等，团队太大，过程中出现的任何一点变化，都需要通知到团队的每个人，沟通成本极高。

当时酒店无线团队采用的是职能型组织架构，这种层层汇报，层层审批的组织架构在当下互联网盛行的时代，尤其是需要充分发挥每个员工创造力的软件行业，不免显得有些笨拙，一个环节出了问题，都有可能影响整个 APP。


酒店无线技术总监
架构组
Service组
APP 组
H5组
测试组
APP Service
H5 Service
Android
IOS
APP 测试
H5 测试
架构师
APP Service
H5 Servic
Android 开发
iOS 开发
H5 开发
APP 测试
H5 测试
技术团队向 PM 批量交付成果
PM1
PM2
PM3
PM4
PM5
PM6
PM7
PM8
PM9
PM10
酒店无线产品总监

2．产品经理说：我们的需求都很重要，没有优先级

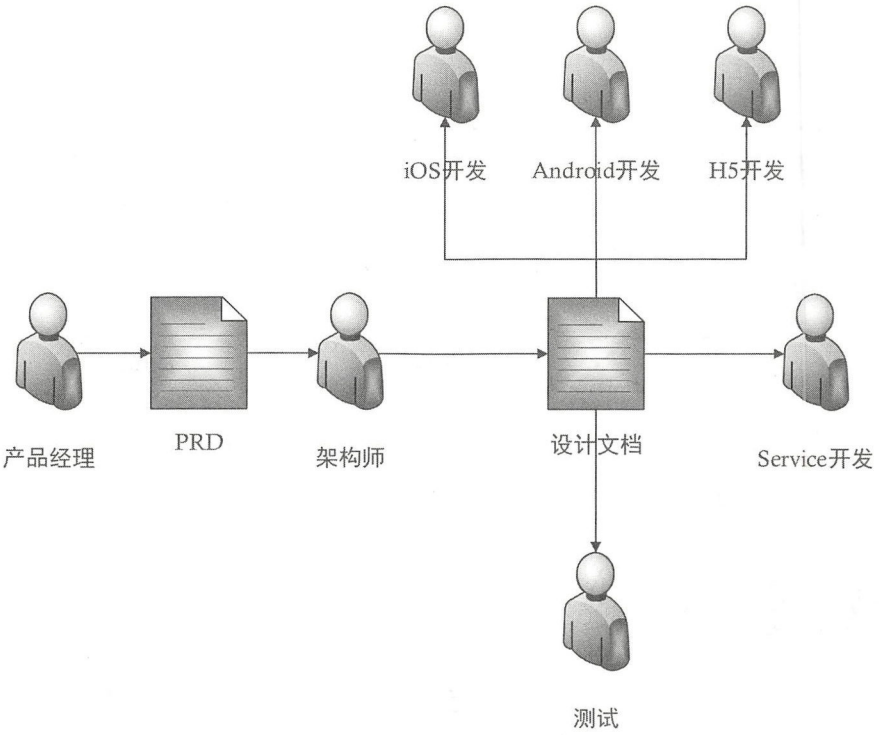
在做转型之前，机票和酒店都是典型的职能型组织架构：Native、Hybrid、H5 三个以技术线划分的团队，产品线耦合在一起，若干产品线在提需求的时候都会用到 3 个技术团队的资源。每次排需求计划会，一旦需求排不上，资源饱和，就会出现产品经理抢资源的情况，产品经理各自强调自己的需求优先级高且非常重要，但实际又没有一个能真正考量所有这些需求优先级的角色，来拍板谁的需求能上谁的不能上。

有时候把老板请来拍板，只会将此刻的场景变得更加尴尬，老板也很为难，到最后经常出现某个产品经理的需求没排上则非常沮丧，或某个开发人员加班加点帮忙产品经理消化需求。

3．技术人员的苦，谁能懂

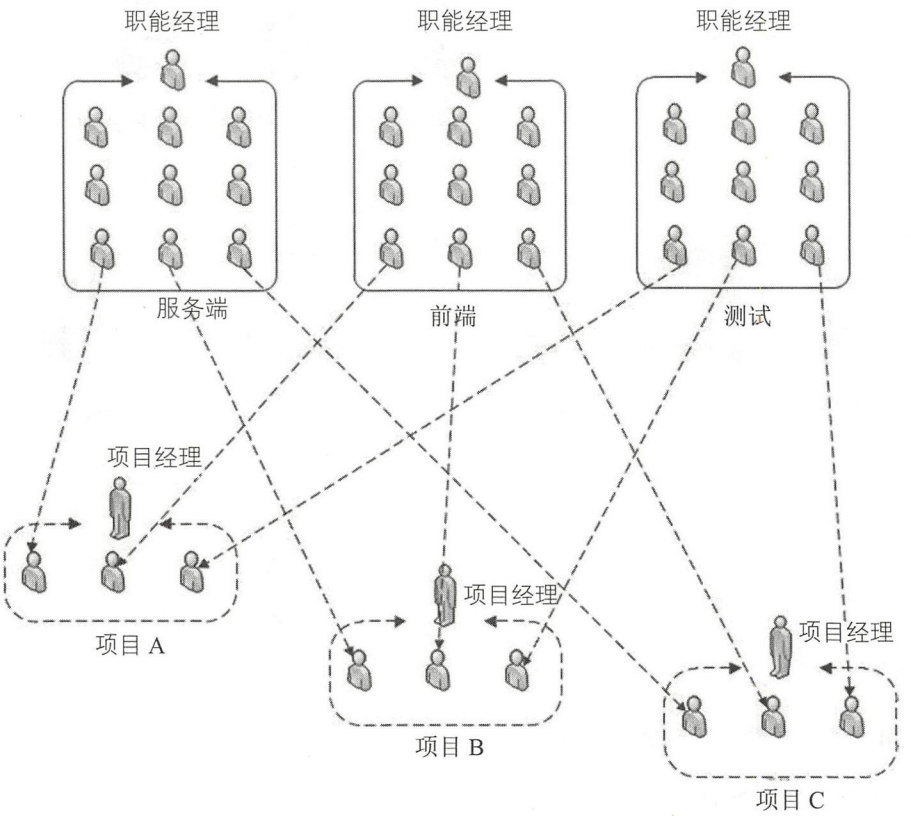
瀑布的开发方式非常不适应需求变化；流程单一，不可逆；风险往往到后期才显露，失去及早纠正的机会。当然这种模式也让架构设计团队（只有 4 人）成为瓶颈，同时架构师觉得架构团队都是做一些纯粹的文档工作，没有技术含量，没有成就感。技术方案是架构师定义的，但是方案的实现却是一线开发人员，这种做法也让开发团队没有参与感和成就感，大家都怨声载道。

一线开发人员几乎完全不懂业务需求，不懂 API，技能相对单一，遇到 Bug 相互推诿，各职能小组都比较封闭，自己的工作不允许别的小组参与。临近发布，才算是真正所谓的大爆发，技术方案变更、产品需求存在漏洞、历史遗留的坑全都涌现出来，大家都神经紧绷，稍微出一点问题，都会争吵不休。尤其集成测试的这一周，每天开发和测试人员都需要疯狂加班，测试人员更是辛苦，经常需要加班到晚上 12 点左右。遇到发布更是恐怖，连续两天通宵也是很正常的情况。频繁的加班以及巨大的压力使团队当时的离职现象很普遍，甚至部分骨干员工离开了团队。



4．项目周期长，老板不满意

机票业务在 2016 年 4 月开始启动订单详情页改版项目，直至 2016 年 10 月流量开至 100%，整个项目周期耗时半年。在排期前，项目拆分了多个小模块，进行技术方案讨论，人员分工，工作量评估和入测时间的排期，但在过程中无法阻止的是大大小小的数次变更和插入的紧急需求，项目排期也调整过多次。整个项目做下来大家普遍感到非常辛苦。熬了大半年上线的项目，得到的数据一开始也没有很乐观，大家心情也都受到了影响。



5．虚拟团队我们也尝试过，均以失败告终

我们也曾组建过类似的虚拟团队，由多个职能团队抽人数组建到各个业务线的虚拟团队中。但其中没有明确职能经理的角色，反而弱化了其管理职能。而实际人员在虚拟团队中还是会经常被调离到其他团队，很多任务都需要在职能经理处转手再做分配。得不到职能经理的支持，虚拟团队形同虚设。

如何转型

1．引入敏捷开发

从敏捷试点开始：各职能团队分别抽取 2 人，组成 Dev 团队，大家按照标准敏捷活动运行，效果显著，对比非 Scrum 团队，这个临时的小团队，大家的沟通合作都有明显改善，问题提前暴露，风险提前预警，很好地解决了团队“最后一公里”的问题。

随即我们邀请敏捷教练培训，走到团队中来指导，协助团队一起发现并解决团队问题。组建 ETC，鼓励团队中的敏捷爱好者加入 ETC，帮助推广敏捷理念。调整迭代周期，由以前的 2 个月 1 个版本，变成 2 周 1 个 Sprint。团队工作透明化，开发过程中各环节保持高度的透明性。开发过程中的各方面频繁地检验，确保能够及时发现过程中的重大偏差，最后就是敏捷标准流程中的几个活动。

	会议内容	输出成果	会议参与人
需求澄清会	1．详细讲解需求； 2．确认本次 Sprint 需求具体的外部依赖，详细至技术方案； 3．估算需求大小； 4．需求拆分至 Story	1．需求大小； 2．具体外部依赖方案； 3．具体 Story	Dev 团队、PO、SM、业务方
计划会议	1.明确本次 Sprint 需求详细技术方案； 2．确认 Sprint 需求范围； 3．开发任务拆分	1. Sprint 目标； 2. 任务 Task	Dev 团队、PM（可选）
Demo 会议	向业务方展示成果		Dev 团队、SM、PO、业务方
回顾会议	回顾已经完成的 Sprint，确定做出什么样的改善可以使接下来的 Sprint 更加高效、更加令人满意，并且工作更快乐	改善的 Action	Dev 团队、SM、PO
每日站会	昨天做了什么，今天的计划以及遇到的问题	Daily 目标	Dev 团队、SM

结合以往团队习惯的工作模式，我们寻找到了最适合团队的会议安排和 Sprint 节奏，保留了 4 大会议：需求澄清会、计划会议、回顾会议、每日站会。由于在测试过程中，测试人员都会主动发起与产品人员进行检查验收，所以我们的演示会省略了。我们 2 周 1 个 Sprint，每周有 1 次发布，保证线上需求和代码的新鲜度。

需求澄清会主要为产品陈述需求，根据优先级讲解需求，会议上各角色评估工时，结合当前 Sprint 的计划投入人力范围内制订出在饱和度范围内的需求列表。当然我们对会议的前置准备工作也是严格要求的，大家务必提前阅读好 PRD，提高会议效率，提问关键问题，评估有效工时。需求澄清会后 TO 会与团队成员沟通，进行需求的分工，鼓励大家自领任务。

计划会议主要将需求进行拆分，大需求拆为 Story，Story 拆为 task 的环节，确保各方对需求和技术方案都理解一致。开发人员根据自领的任务量，结合需求优先级，与测试人员沟通好入测时间，这样，一个完整的 Sprint Backlog 就确定好了。

每日站会是整个团队每天固定时间的碰头会，站会的形式也变革过几次，之前的形式是根据需求条目的顺序 SM 问大家答，后续由于出现不知道部分人每天的工作情况，调整为大家各自主动来说自己的完成情况和碰到的障碍。为提高大家的主动性，看板也由之前的电子看板调整为了实体看板，每个人各自去拖动自己的任务条，不再由 SM 去更新进度。

回顾会议我们一般是在 Sprint 结束后的第二天召开，Sprint 刚刚结束，过程中的问题大家都还比较深刻，我们及时总结，碰到的问题都会记录下来并安排对应的 Owner 去跟进，在下一次回顾的时候追踪这些问题的状态。

2．大团队拆分为小团队，小团队固定产品线

我们在组织架构上也做了相应调整，由以前职能型架构变成跨职能特性团队。我们将原有的“无线大团队”拆分成 6 个“无线小团队”，每个 Scrum Team 都有 Android、iOS、服务端以及测试的人员，大家以 Scrum Team 为单位，位置调整在一起，面对面沟通。每个 Scrum Team 有固定的产品线，关注点更加聚焦，同时设定产品目标，Team 内的每个成员都要对自己团队的目标负责并为之努力。

每个 Scrum 团队都配备了产品 PO，即产出 Scrum 团队产品优先级，为用户和客户创造优质的产品来实现商业目标的角色。团队里会有多个 PM，PM 产出与该业务线相关的 PRD 和需求跟进，但整体业务线的优先级都是 PO 说了算。

配备了 SM（Scrum Master），负责流程上的成功，帮助 PO 和团队用正确的流程创建成功的产品，是为推进组织变革和建立敏捷工作模式提供帮助的角色，由我们团队之前的项目经理转型代之。

配备了 TO（Technolog Owner），负责团队的技术改进，质量提升，把控整个团队的技术方向，由我们团队之前的职能经理或架构师担任。

每个团队都配备了固定的开发和测试人员，APP 的 3 个 Scrum 团队则同时配备了 iOS 和 Android 开发人员。大家位置都调整坐在了一起，面对面沟通，有什么事大家吼一声，能当面沟通解决的事情，不再用邮件处理。

团队有了，团队目标也有了，大约两个月后，整个团队有了明显的改善：以前以邮件为主要通讯的方式不见了，能说话表达清楚的就不再打字发邮件，能当面说清楚的 Bug，就不再提 Jira，团队内有任何需求或者技术方案的变更，只要吼一声，大家就知晓，保持信息同步，大大降低了沟通成本，也解决了之前邮件沟通不当带来的误解。我们坚持实用主义，共同探索能实现团队目标的最好的团队合作方式。

3．技术带动生产力的提高

不管怎样，4 个团队开发的还是同一个 APP，过程中难免存在一定的冲突，如何解决冲突，不影响小团队的效率呢？我们搭建多套测试环境，各团队使用自己独立的环境，互不影响；APP 与 SVC 之间的协议采用 PB 模式，减少序列化和反序列化失败；搭建自动化测试平台，减缓测试压力；推进 Code Review 以及 Sonar 落地，强化代码规范；另外还开发部分工具，方便开发人员定位问题，如支付 Mock 工具、SOA 报文验证平台、SoapUI 工具、Redis 工具等。

我们相信技术带动生产力，今后我们也将继续加强敏捷工程实践，搭建自动化构建平台，持续集成。

敏捷转型成果及团队成员评价

从结果上来看，各团队都在自己的产品领域有了不同提升。

从团队建设上来看，我们采访了一线开发人员以及测试人员、产品经理、职能经理，他们分别谈了自己工作的变化以及团队这一年多的转变，相信他们是最好的见证者。

架构师说：现在有更多时间去关注系统的稳定及性能，由原来单一的“传话员”，变得自主去思考，发掘系统中的隐患，更多关注系统的优化和改造。

职能经理说：服务端人员对业务需求理解得更加透彻，很少有返工的现象，需求开发速度也比之前快很多，开发人员打破了工作边界，跟产品经理和测试人员都有很好的沟通和配合。敏捷改革之后，也暴露出很多团队的问题，大家一起解决，一起改进，相信团队会越来越好。

iOS 开发甲说：以前是一群人一起开会，满屋子黑压压的，全是人，到开发的时候，开发的内容跟 PRD 上描述的差别相当大，上线需求经常被砍，做完的功能还要回退。现在开会的人少了，规模小了，会议内容明确了，端对端直接沟通，做好的功能都能及时上线，产品的需求也没以前变化那么频繁了，大家合作越来越紧密了，很开心！

iOS 开发乙说：一年前整个团队都围绕着一个需求池，很乱，很笼统，有一个很大但是不明确的方向，虽然产品经理负责各自的产品线，但是产品还是有很多不明确的地方，需求变更太频繁。但是现在通过需求澄清会之后大概会知道产品的需求点，在计划会议之前整个团队会对自己团队需求做个方案调研，计划会议后，团队内的所有成员都会对需求的方案有所了解，需求变

更和方案变更都大大减少，开发质量也有明显提高，整个团队的沟通也大大增强了，大家合作得比较愉快。

产品经理说：现在大家都是“一条船”上的人，同呼吸，共进退！

测试人员说：以前 Bug 都要非常详细地在 CP4 中写上复现过程，经常几天 Bug 修复、都没有进展，非常难以推动，现在大家以团队为单位了，只要是自己团队的问题，大家都积极解决。不仅如此，遇到测试工作量比较大的时候，开发人员会帮助我们一起进行测试，减缓我们的压力，感谢团队！

建立高效的团队，成为最好的自己，拥抱变化，勇于创新，我们一直都在路上！

电影《功夫》里火云邪神的经典台词“天下武功，唯快不破”，意思是：天下没有哪一种武功是不能被拆招的，即没有最强的武功，唯有速度快，对手根本来不及反应，你先发制人，在对手还没有出招前，就将对手击败，对手何以拆招，所以唯快不破。如何让我们的团队更快呢？拆分是第一步，接下来就是如何从我们的沟通、协作、速度、学习、成长等各方面来管理我们的小团队，让其发挥更大的价值！

2.3　天下武功，唯快不破

2.3.1　沟通快——开会那些事儿

众所周知，Scrum 最典型的就是会议，除了 15 分钟的每日站会，还有需求澄清会，计划会议，Demo 会议和回顾会议。会议围绕在我们身边，很多人都会觉得会议太多，然而任何一个会议，都有它存在的必要，所以高效的会议对我们而言，是多么的重要！

每日站会

每日站会是一种限定在 15 分钟之内的活动，目的是让开发团队同步行动，并且为接下来的 24 小时建立计划。具体的做法是：检视上一次站会后所做的工作，然后预测下一次开会前可以做的工作。为了降低复杂性，站会在每一天的相同时间以及相同地点召开。在会上，开发团队成员做如下说明：昨天做了什么，今天的计划，遇到的问题。

每日站会是实施 Scrum 后第一个会用到的会议模式，但是经常演变成团队成员一一向 Scrum Master 汇报工作，线下能及时沟通的问题也都放到站会上沟通，慢慢地就变成了汇报会议，对于项目整体的风险和状态并不关心。

碰到以上问题可以尝试以下改良版本的经典三问：昨天的目标是否完成？今天准备做什么？目标没有完成碰到的障碍是什么？每个人针对以上三个问题做针对性的回答，既明确了目标又加强了团队成员之间的交流和信息共享。它是一个检视迭代进度并对迭代计划进行实时调整的重要会议。

提高站会效率可以帮助每个成员都了解项目的状态和风险，明白自己的工作对项目状态的影响，及早暴露风险并及时解决。

需求澄清会和计划会议

需求澄清会和计划会议是大家平时比较重视的会议，但是一般都被大家理解为故事估点、拆解任务、制订提测时间的一个过程。需求澄清会和计划会议的目的分别是明确做什么，怎么做。

会议的安排可以一起进行，也可以分开进行。对于逻辑性不复杂，团队可以当场根据经验设计方案的需求，就不需要额外举行计划会议；对于业务复杂，依赖项多，无法当场评估大致实现方案的需求，可以分开进行，给予研发人员进行前期调研的时间。对于每个需求的澄清，不一定讨论到像素字号，但一定要明确这个需求的背景是什么，做这个需求的意义是什么，是为了解决什么问题。

举个例子：关于字数过长的显示逻辑，在未进行团队一致确认的前提下，可能会经历以下过程。

1．开发 A 进行了截断处理。

2．测试 A 发现后提出一个 Bug，期望执行显示。

3．开发 B 接手解决 Bug，根据测试的预期修改了显示逻辑。

4．测试 A 验证通过，产品回顾失败，需要调整为截断。

5．再把代码逻辑重新调整回来。

这样的一个过程在我们平时的需求开发阶段比比皆是，前期的需求未明确导致后

期沟通成本变高。对于 Sprint 周期内做什么，怎么做，团队有明确的方向后，在过程中重复确认，低效沟通的成本就大大地节约了下来，有效地保证了 Sprint 周期内需求的有序发展。

回顾会议

回顾会议是一个比每日站会更容易被忽视的一个会议，包括 Scrum Master 新手。回顾会议通常在 Sprint 结束后大家回忆周期内哪些是做得好的，哪些是需要改进的。会议结束可能会迸发出超级多的改进点，但是无最终的落地方案，久而久之，回顾会议演变成抱怨会，团队成员对回顾会议的信任度急速降低，甚至抵触出席该会议。

碰到以上问题，Scrum Master 需要敏捷自己，观察团队，针对团队暴露的问题精简出最重要的 1 到 2 个改进点，让团队成员仅围绕相应问题思考并落地改进方案。接下来的周期内观察落地方案是否得到充分的执行，并总结是否还有更棒的方案，持续改进得到一个最适合团队的方法。团队问题得到解决后，团队成员对于回顾会议的信任度也会逐渐加强。大家踊跃参与，团队配合度也会得到一定程度的提高。

它是一个持续不断改进的会议，通过持续不断的改进成就一个不断成长进步的团队。

如果团队里还是不时地有人抱怨会议太多，每天都开会都没有时间做事情了，这个时候我们一定要停下来，审视下团队，我们是否明白每个会议的目的，是否都有针对性地进行会议的开展，是否都得到了相应的会议结果。针对问题，进行剖析，持续改进。

2.3.2　协作快——协力致胜，团队为赢

部门之前一直是采用传统的瀑布流式的开发模式，产品经理设计需求，之后对需求评审，然后估算工期，排计划，最后执行。最近互联网公司都流行搞敏捷开发，所以我们也开始尝试，但是操作起来总是感觉困难重重。然后我找了业界比较资深的一名敏捷教练，希望请她帮忙。

教练来到我们团队后，开始是按照一贯的方法，所有敏捷类书上都会提到，帮助团队搭建看板、开站会，看板一直持续更新，站会也坚持下来了，却发现一个问题，

几乎所有的领导一致认为站会是没有意义的。这些领导曾经尝试过，最后发现每日沟通花费了时间，但却并不会给他们的工作带来多大的帮助，于是都放弃了。

为什么在那么多团队行之有效的实践，在我的团队就推不下去呢?

经过教练和我的调研，我们找到了问题的答案——要找对需要在一起沟通的人，这比沟通的形式更重要。

我们专车的团队是个典型的 APP 开发团队，分为 iOS、Android 和服务端开发三个组，维护着两个主要的产品，分别是给用户下单用的用户端和供司机抢单用的司机端。

当我们按照组织结构召集一个团队开会的时候，团队成员之间由于工作的相关度不高，所以并不太关心其他人干了什么，虽然站会时间不长，可大家觉得没有价值，是在浪费时间。而当我们将团队重新组合之后，按照产品线划分成司机端、用户端和纯服务端三个团队，每个团队都包括 PM、各种 DEV 以及 QA。

调整完之后，大家沟通的积极性马上就高了起来，因为大家共同在做一件事情，是上下游的关系，相互能够给对方提供信息并解决问题。最终我们其实只做了一件事情，就是建立了一个完整的交付团队，也就是通常人们说的“敏捷团队”，所有的实践就顺理成章地实施起来了。

从这里我们可以看到，敏捷的团队比敏捷的流程或者实践更重要。那么，什么样的团队才能算是一个敏捷的团队呢? Mike Cohn 给出过 9 个问题，用来判断一个团队是否“结构优良”，我也认真研读过相关的资料，这里给出几点我认为最重要的特点。

1．要有交付能力。能够独立交付完整的业务功能，这是一个敏捷团队的最基本的特性。这就意味着团队必须包含交付所需要的全部角色（一般至少要有前后端开发人员、测试人员），或者具备全部角色能力的人员。

2．要“高内聚，低耦合”。团队所承接的大部分需求都应该能在团队内部完成，对其他团队的依赖应该是少量的或者简单的，同时，应该尽量避免两个以上的团队共享一个成员的情况。这样才能进一步地减少等待以及各种管理的浪费，保证快速交付的能力。

3．要保持小团队。团队太大会降低沟通效率，增加管理的浪费。而小团队则更容易保持对目标的专注并形成凝聚力。一般认为 7 加减 2 是比较合适的范围。

4．要足够长期。团队需要一定的时间去形成统一的规则和文化。一个新团队的磨合期通常至少需要 2~3 个月的时间，而要想在这个基础上形成自组织、自管理的团队文化，则需要更长的时间。只有成熟度非常高的组织（团队文化和基础设施都高度成熟），才能频繁地变更团队的组成而不破坏团队文化。

这个改变过程也让我进行了深刻的反思，我们从一个纯粹不敏捷的团队到中等程度敏捷实践的团队，从改进的角度来看，属于渐进式优化的做法：在保持组织结构不变的基础上，按照业务和产品的结构划分出“虚拟的”交付团队。在这个团队里，各种角色能够频繁地、一致地、充分地交流，及时发现和解决问题，从而快速交付产品功能，实现业务价值。

我们和其他团队进行了沟通交流，发现目前按照我们这样成功划分虚拟交付团队的部门并不多，主要的原因是很多开发、测试人员没有与产品经理形成对应的分拆，无法形成完整的交付团队。

2.3.3　速度快——自动化让团队飞

在敏捷研发的模式下，敏捷的测试当然是非常重要的一环，很大程度上影响着敏捷研发的成功实施。那么，测试如何做到相应的敏捷，哪些是需要重点关注的呢？下面从三个方面阐述个人观点：

测试团队及人员

Scrum 团队可以以实体和虚拟两种组织形态存在，多数情况下，Scrum 团队成员都是从各个角色资源团队抽取组建而成的一个虚拟团队，有些团队组建形成后便长期存在，有些则根据业务实施情况不断重构。因此，角色资源团队和 Scrum 虚拟团队是一种矩阵关系，Scrum 中研发团队的成员则是分别由多个组建开发团队和测试团队抽调组成的。

对于测试团队，成员可能分散在不同的 Scrum 团队中，在一定程度上测试成员的工作安排和使用都受控于 Scrum 团队。由于每个 Scrum 团队的情况和需求都不一定足够一致，那么作为测试团队则需要一个公共小组为各个 Scrum 团队提供有力的技术基础服务和技术落地的管理工作，例如提供基准的测试流程规范、通用的测试平台和框

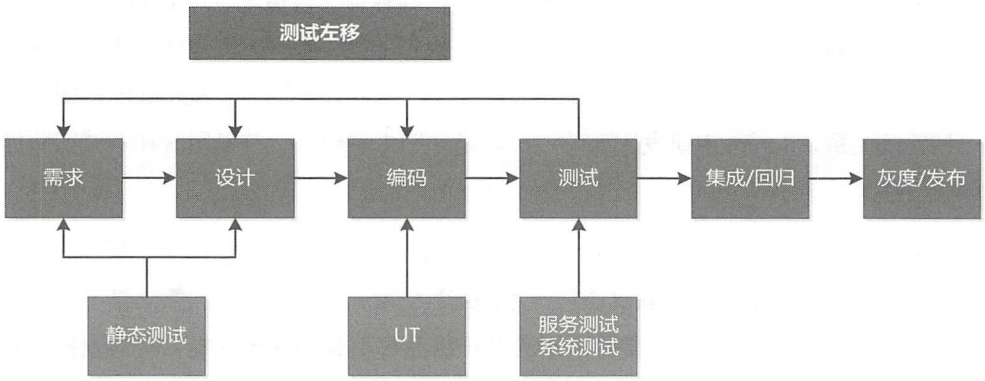
架、自动化测试技术支持、个性化测试工具支持等。而作为 Scrum 团队中的测试成员，由于 Scrum 团队更多地强调人员简而精，则他们需要具体良好的综合素质，除了基本的业务能力、独立的功能测试能力，还需要具有一定的编码能力、自动化测试开发能力以及较强 Trouble Shooting 能力。

总之，测试团队为 Scrum 测试成员提供统一的基础规范保障和必要的技术支持，确保每个 Scrum Team 的质量保证能力在一个水平线上，而不是完全依赖成员的个人能力。而 Scrum Team 成员具备在总体规范的基础上培养较强的单兵作战能力，能够在背靠测试团队的技术支持下灵活应对实际研发过程中的各项任务挑战。

质量保证流程

虽然测试团队的绝大多数资源都会分配到各个 Scrum 团队中，但是测试团队还是需要为每个 Scrum 团队提供一套统一的基础流程规范。测试作为敏捷项目研发中重要的一环，为了避免项目不会受到质量问题影响业务目标而降低敏捷度，那么测试的工作就不能在最后的环节才实施，而是努力做到全员和全程质量保证的流程体系。这一点很多企业的技术团队都提出了测试前置的概念。

测试前置也可称为测试左移，它并不是单纯意义上的让测试人员和测试工作更早地介入，而是建立一整套全流程和全员质量保证的理念，这些工作更多地需要测试人员去驱动和引导。其中涉及的工作大致可分为 PRD 静态测试、服务契约测试、代码静态扫描、单元测试、服务接口测试、开发自测、测试准入检查、入测质量数据统计等。



测试前置的工作初期可能会造成测试和开发人员的工作量加大，但是如果流程逐

步顺畅、产品设计质量和代码质量逐步提升，进入良性循环后，研发整体的效率和过程中的工时浪费将得到明显改善。

推进测试前置工作中需要把握几个重点：

1．作为测试团队的负责人，需要得到上级领导的认可与支持，制订出可行方案并由上而下推行，同时得到平行的开发团队和产品团队的认同与配合。

2．产品经理、开发人员、测试人员等角色中涉及测试前置的工作，需要评估工作量，例如 Scrum 工作模式下，Sprint 中每个 Story 涉及的 UT、开发自测、产品自测、测试验收等工作耗时都需要计入工作量，否则即使全员有质量意识但在高强度的工作下也可能有心无力。当然前期可能对 Scrum Team 的任务吞吐量造成一些影响，但是从长远看是有利于质量和效率的提升的。

3．产品的 PRD 和开发人员的代码在结构上均需要优化可测试性。产品文档主要是在格式和逻辑梳理上方便进行场景分类测试。

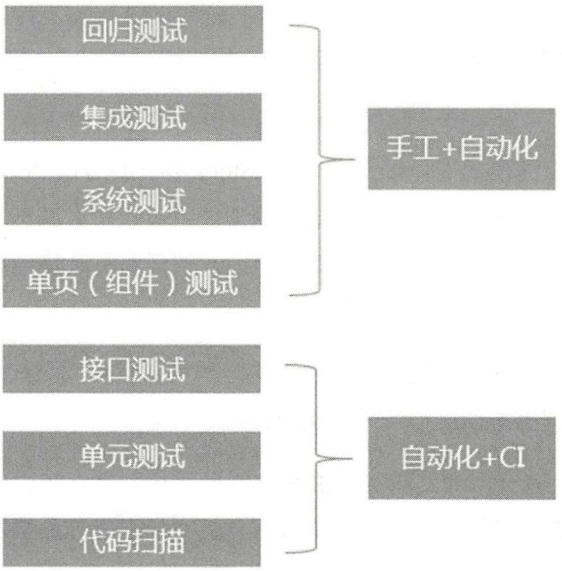
4．技术层面上需要为测试前置提供有力的框架和工具支持，例如低门槛、高效率的自动化测试框架，持续交付、持续集成平台，方便开发产品自测的测试数据构造工具等。

5．测试人员在这个过程需要扮演串联各个环节和监督、总结的角色（当然 Scrum Master 也可以承担其中一些工作）。每个 Sprint 结束回归会议上，测试人员需要提供每个节点各个角色的工作数据，例如代码静态扫描的问题及改善、UT 的覆盖率、开发的自测覆盖率、自测通过率、入测准时率等。这些客观数据用于鼓励团队成员持续改善我们的前置测试工作，并且帮助发现 Sprint 过程中的问题点。这项工作非常重要，数据的客观和准确需要得到每一位成员的认同，只有这样才能将这项工作长久持续坚持并且优化好。

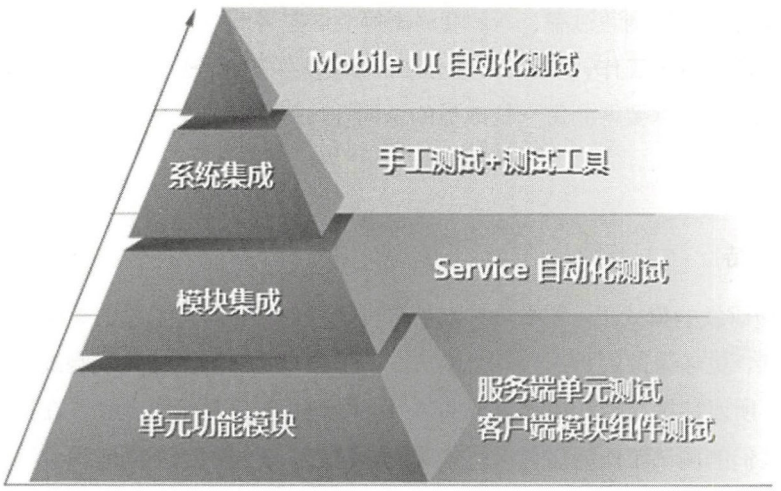
测试技术支持

在敏捷研发的模式下，需求往往被拆分得较为细致，同时研发的编码和测试时间都相对紧张，更多采取的是迭代并行的方式。那么，回归测试和关联测试的工作量势必会增加，所以持续交付和持续集成的体系必须建立起来并给予支持。在持续集成的基础上，我们的测试工作就需要进行分层，分散在研发过程的不同阶段。我们所在的

手机端研发团队，目前大概可分为服务端单元测试、客户端单元测试、客户端组件测试、服务端接口测试、功能模块集成测试、系统测试、回归测试、预发布常规测试等。具体的测试方式包含手工测试、自动化测试、随机测试等。



关于各个层面的测试占用总测试的比重，每个团队需要根据自身的业务和技术情况而定，整体的原则是：越下层的自动化测试所占的比重越大。



自动化测试在执行层面上，则需要考虑一些周边的配套设施的建设，以便自动化测试能够稳健地运行和管理，其中涉及系统之间和服务之间的依赖、测试环境的稳定性、测试数据的构造、自动化校验方式等。

在敏捷研发模式下，测试基于风险测试的同时，要兼顾质量和效率的双保障，那么自动化测试等技术的应用则是势在必行的。自动化测试并非单一的技术个体，它分布于系统架构的各个层面，覆盖白盒测试、黑盒测试、灰盒测试等多种测试方式，更重要的是它需要全方位的配套体系的支持，包含且不局限于测试前的测试数据、测试用例的自动化构造、测试环境的自动化搭建、后台依赖的隔离，测试中的自动化运行管理、个性化的校验方式，测试后的数据还原、恢复、测试结果聚合报告、排错等。这些都属于测试自动化体系中的多个重要环节，每个环境协同配合好才能将自动化测试工作顺畅、健壮、低成本地运行起来。总之，将全方位的质量保证做好是敏捷的重要前提之一。

2.3.4　学习快——组建学习型团队

资源瓶颈应该是很多团队都会遇到的问题，产品经理在提需求的时候，更多的是考虑需求的价值，而非资源平衡问题，团队也很难做到完美的资源平衡，所以全栈就成为了我们一个重要事宜。如何建设全栈团队，第一步就是组建学习型团队。大部分人在看到这个标题时，马上想到的就是各种培训，然而我们想讲的是：学习的 721 理论。

70%来自于工作中的经验积累（工作方法+经验技巧）；

20%来自于人际交往、沟通交流；

10%来自于培训课堂。

学习型团队是一个能熟练地获取和分享知识的团队，同时善于自我修正、改进，以适应环境挑战新的工作目标。建立浓厚的学习氛围，定期设置目标，通过多种方法相结合进行系统学习。团队成员坦诚、融洽、互助，信息和知识能快速地互相传递，好的经验能够快速应用，提升团队成员个人技能，促进团队的发展。

分享学习

集大家之精华，为了挖掘出各团队比较好的经验，我们组建了分享委员会，主要是从各职能角色中，寻找出一些比较专业的人员，大家一起来制定相关的课程，明确学习群体的需求。对于一些丝毫没有概念、第一次接触的同事来说，我们必须从入门开始，系统介绍新的知识；对于一些做过一两年相关工作的同事来说，我们注重的是日常工作中遇到的问题，经验的分享，各种办法尝试的成果。明确学习群体，制订学习目标，这样才让大家更有收获！

很多人认为，打造学习型团队就是培训和分享，没有什么新鲜的。当然培训和分享其实仅仅起到抛砖引玉的作用，从外部支援的角度为打造学习型团队提供理论上的解释和操作上的咨询，其本身并不是打造学习型团队的必经环节，更不是学习型团队的本质意义。最好的学习就是在实践中去做，也许你什么都不会，但是拿到一个用户故事，边学习边做，边学习边找身边的牛人请教，这一定比你听各种讲座，上各种速成班要有效得多。学习不能停留在表面上，而是要将学习到的东西融入到我们自己的工作中，真正地做到学以致用。

找搭档

实践出真知，万事开头难，从 0 开始去学习，都会困难重重，如何更好地进入状态，不至于做到中途就放弃呢？找搭档是一个非常关键的点！你可以在团队中，找到与自己比较合得来，并且技术能力不错，且愿意教授他人知识的搭档，这个人将在你整个的学习过程中，为你提供很多便捷的帮助！当然师父领进门，修行在个人，一切开始需要靠自己，但是这个搭档将会作为路灯引领你向前。

竞赛

如何刺激大家去学习新知识呢？我们能想到的最好办法就是竞赛！今年我们举办了一系列的竞赛，当然并不是那种空洞的大赛，而是简单地把我们日常工作中遇到的难解问题拿出来，大家线下组队，不同职能角色的人员组成一个团队，随机抽取题目，一起想办法来攻克。分享委员会的同事们担任嘉宾，最终决出比赛的一等奖、二等奖、三等奖，以此来激发大家的学习动力。

当今世界发展如此之快，尤其是 IT 行业，各种新型技术层出不穷，要想跟上发

展的步伐，快速学习显得尤为重要！创建学习型团队是一个漫长的、艰苦的过程，必须结合本企业的实际情况，不断探索，不断总结，以期望建立起具有自身鲜明特色的学习型组织，真正促进企业的长远发展。

学习使人进步，使团队保持生机和活力，使团队在当今激烈的竞争中保持不败！

2.3.5　成长快——敏捷开发的持续改进与变革

经常会有人问："我们应该怎样开始做敏捷？"或者"能不能来帮我们推一下敏捷？"这种问题我通常都不敢轻易回答，敏捷有很多实践，管理的、工程的都有，但敏捷绝非我们看到的站会、持续集成、TDD 等那么简单，真正的敏捷体系是从理念到文化的一次变革。所以具体到一个团队，究竟为什么要做敏捷，能够多大程度地承受和改变所带来的痛苦和风险，本质上还是需要自己先想清楚，我们要解决的问题或者期望的价值究竟是什么，再来判断应该做什么以及怎么做。

这里分享几个敏捷相关的过程改进案例，希望能够给大家一些可以借鉴的东西。

案例一：灵活地响应变化

两年前我在酒店事业部做支持的时候，有一次与一位产品总监聊天，他向我抱怨："技术团队怎么有那么多的项目延期？他们能不能靠谱一点，至少在我的周计划里，承诺要完成的事情应该做到吧！"

我反问了一个问题："那作为一名产品总监，你能不能保持你的团队一周内的需求不改变呢？"

他想了想说："我做不到。"

我们相视一笑，很容易就达成了共识——我们不能指望不变或者少变，而是要提升整个团队的应变能力，去响应甚至拥抱变化。

那么，怎样才能提升团队响应变化的能力呢？

首先，要建立良好的可视性。能够清楚地知道每个人每天都在做什么，进展如何，何时能完成。这样在发生变化时，我们才知道应该安排谁、什么时候去做是最合适的。所以我们做了 Story/Task 的拆分，并搭建看板（如下图所示），展示每个人的工作，再通过站会沟通进展以及问题，保证信息每天都能得到更新。

其次，要提升变更决策的效率。我们要让有价值的变更能够快速地得到实施，要有个快速且有效的决策机制。那么，应该由谁来做这个决策呢？决策人级别太高，决策效率会低；决策人级别太低，又担心水平不够，不能服众。我的观点是“让具备这个能力的、级别最低的人来做”——资深人员能做就不要让部门主管做，部门主管能做就不要让总监做，总监能做就不要让 VP 做。有做得不对或者不好的时候可以升级问题，但不要一开始就把职责都定在管理者身上。

所以我们引入了 PO（Product Owner）的概念。为每个团队指定一个 PO，他负责接收所有的需求，判断价值和优先级，如果有需要的话，还要帮助其他产品经理做系统的需求分析。大部分团队的 PO 都是资深的产品经理，但也有小部分的技术团队，由于对应多个需求方，最后技术主管自己做了 PO。在实践中我们发现资深的产品经理和技术主管都能把 PO 这个角色做得很好，需要升级到管理层决策的情况非常少。

最后，要降低变更管理的成本。变更过程中最令人感到痛苦的事情就是 Re-Plan。一般来说，如果只是调整当前项目的计划，还相对比较容易，但由于项目延期或者对人员的时间占用增加，通常会导致其他项目的等待或延期，这种情况沟通起来就比较费时费劲了。更糟糕的是，技术团队已经对计划做出了承诺，客户就会很愤怒：“你们这帮人怎么这么不靠谱，承诺的事情为什么老是变！”所以很多技术主管会倾向于拒绝变更，他们会说：“你们先去走个流程！”“你们先去找 VP 审批一下！”

所以我们转变做计划的思路，从承诺每件事转向只承诺当前优先级最高的事情。从“事推动人”转向“人拉动事”，也就是说我们尽量让每个人都工作在优先级最高的事情上。做完一件事情，再去需求队列里获取当前优先级最高的事情，在完成之后，再去取下一件事情……如此循环。这样我们就把对变更的决策与制定工作计划这两件事情“解耦”了，让整个团队都能够聚焦在变更所带来的价值上，而不是痛苦和折腾的过程。

关于如何制订工作计划（就是我们通常说的“排期”），这里推荐一个实践：每天上午站会结束之后，会有一个小型的计划会议（Planning Meeting）。如果前一天有人交付了手头的工作，就留下来参加计划会议，主管会综合待排期的需求优先级、当前人员的能力等因素安排他们的工作。原则上已经开始开发的工作尽量不要中断，除非有特别紧急的事情，主管会暂停部分相对低优先级的工作，抽调人手响应紧急情况。一个 10 人以内的团队，平均每天花费的时间不会超过 15 分钟。

这个案例是最轻量的敏捷实践，在不改变组织结构和工程流程的前提下，能够以最低的成本实现灵活的应变能力。这也是目前在金融事业部和大住宿事业部绝大多数团队所采用的实践。

案例二：目标一致的团队

> 去年下半年，在一个部门做支持的同事跟我反馈，说她觉得团队的流程在退化：原本分拆了的 FE 和 QA 团队又合回去了，团队从原本的每日排期变回了周排期。跨团队的项目越来越多，沟通、协调的过程也越来越复杂。整体的可视性在下降，管理的浪费在增加，当前虽然还没有明显的问题，但长期发展下去团队的整体交付能力会下降。
>
> 当时我们的 CTO 正好坐在我旁边，问了几个问题：“如果一个流程是好的，它就不应该会退化。如果它退化了，是不是说明有什么地方是不够好的？退化的原因究竟是什么？”

为什么有些团队的流程在过程支持的同事退出之后，就会慢慢退化呢？究竟是因为人的惰性，还是流程的设计本身就有不合理或者不到位的地方呢？

经过对已经实施的团队流程的反复检视，我认为流程会退化的主要原因有两个：

一是这种虚拟团队的组建机制不够强壮。这种虚拟团队之所以能够形成，本质上是基于所有的资源经理的承诺，这是一个不太稳固的基础。尤其是当资源团队的

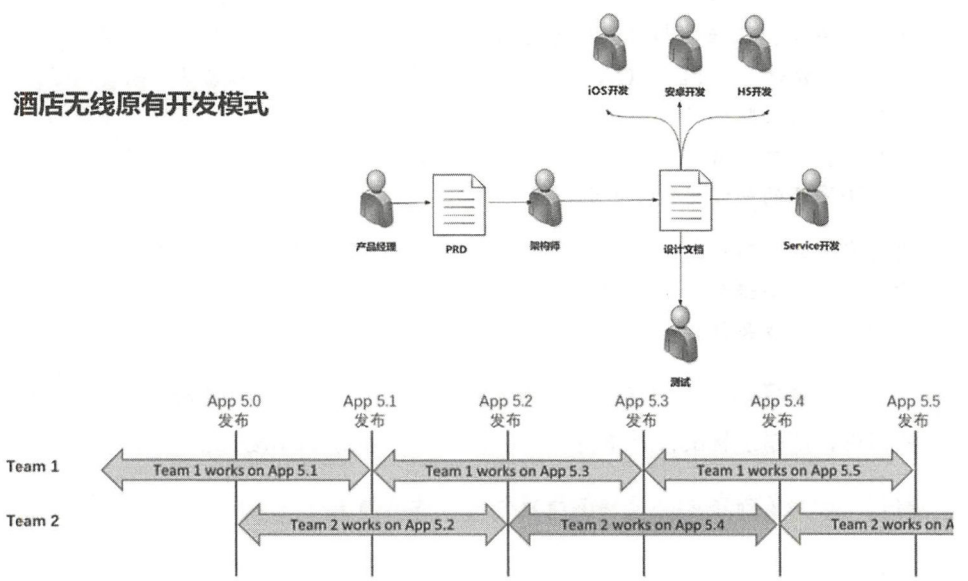
Leader 发生变动的时候，新的 Leader 没有经历过改进的过程，不理解为什么要这样做，或者不知道这种情况下该怎么做，通常都会选择自己最熟悉的管理方式。这个时候，如果团队中没有一个真正懂得这套体系的价值，并且能够强势坚持的人，流程就会发生退化。

二是团队的职责范围定得太窄。我们的交付团队最初建设的时候，基本上是基于一个系统的，比如酒店交易的“用户系统”、“订单系统”。这在产品/系统建设的初期是没有问题的，因为每个系统本身都还不完善，需要做的事情很多，大部分需求都可以由一个团队独立完成。但是随着产品/系统的建设成熟，单个系统上可以做的东西越来越少，更多的需求会需要跨多个系统修改，这时候团队的“高内聚，低耦合”特性就不存在了。

那么，怎样解决这样的问题呢？

2017 年 3 月中旬，我去上海和携程的技术与管理同事们做交流，有幸与大家公认的“敏捷做得最好的团队”——酒店无线的技术团队，做了一次关于敏捷实施的交流。在这里把他们的经验分享给大家。

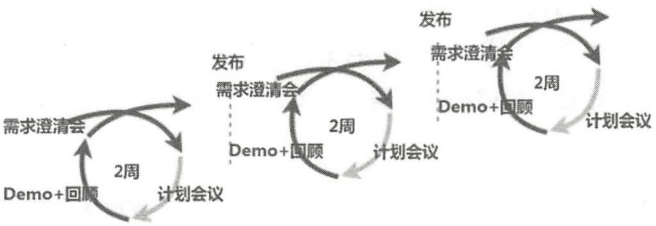
酒店无线的技术团队有 70 多人，组织架构按照技能划分为 iOS 团队、Android 团队、Service 团队和测试团队（见下图），存在的主要问题包括：



- 开发周期过长，从拿到 PRD 到交付要跨越 1 个月或者 2 个月。
- 需求变更频繁，2 个月内会有很多变化，导致已经开发的代码被浪费。
- 团队成员技能单一，不能根据需要互相援助。
- 开发团队只是被当作执行者，没有目标感和参与感。

最终这个团队选择了 Scrum 的模型，将原有的组织结构整合重编，调整成如上图所示的一系列的 Scrum Team。每个 Scrum Team 都是完整的交付团队，包含 iOS、Android、Service 开发和测试。并且鼓励一专多能，iOS 开发人员也会参与 Service 的开发工作。每个团队分别对应不同的目标（OKR），比如：用户/订单增长、基础体验优化、系统架构优化等。

酒店无线现在开发模式



- **开发周期调整为两周一个Sprint**
- **重组大团队为多个跨职能小团队，每个团队12人左右**

经过这样的调整，大大缩减了沟通和管理的成本，提升了工作效率。团队聚焦于目标的实现，会更加积极地参与目标的制订，团队的士气也得到了提高。

这种做法不仅解决了案例二中组织结构不稳固和职责范围过窄的问题，更好的地方在于，它把每个团队的目标和特定的业务目标明确地关联在一起，使得每个团队成员都有可能最大程度地发挥主动性，帮助业务更好地取得成功。我们每年举办诸如 Hackathon 之类的活动，希望工程师们能够发挥创意，解决更多的业务问题。而事实上，如果能够提供更加有效的组织结构和管理流程，工程师们每天都可以 Hackathon。

这个案例是局部的比较彻底的敏捷实践，采用了组织变革的方式，将组织结构调整为基于交付团队（敏捷团队）而非职能团队。并且在这个基础上，逐步形成 “无边界”的工程师文化。这个实践目前在携程的酒店无线技术团队尝试，我们期待这个

团队能够成为一个标杆，带动大家走向更加开放、更加敏捷的文化。

案例三：强健的工程体系和文化

采用 Scrum 的这种模式，显然在速度和灵活性方面有很大的优势。但是，天下没有白吃的午餐，想要获得好处，就必然要付出代价。那么，敏捷要付出的代价是什么呢？

显而易见的，首先是资源问题。敏捷团队建设过程中最常见的情形之一，就是资源经理说“我的人手不够，需要在不同的开发组之间协调资源，不然不够用。”“如果要拆下去的话，我们需要多招 *N* 个人，还是现在这样效率高。”

所以这里就涉及人的能力问题，一方面我们需要有招聘人的能力，以达到各种资源的合理配比；另一方面，我们要有培养人的能力，鼓励团队成员成为“多面手”，每个人都能承担至少 2 种角色，这样在有需要的时候可以在团队内部相互弥补。事实上，资源问题是个比较容易解决的问题，只要大家有意愿、去努力，几乎所有的团队都能做到。

而真正严重的，其实是质量问题。很多强硬地推行了敏捷最后又退化的团队，绝大部分都是因为控制不住质量。网上有很多这样的案例，大家可以搜索着看。

去哪儿酒店以前的交易系统 HMS，当时就是有 4 个独立的开发团队在上面工作，分别对应不同的业务，就是因为没有控制住质量，最后不得不用上 100 多个人，花几个月的时间，把系统重新做了一遍，就是现在的 QTA。以至于团队的老成员们都有点心理阴影，一听说要让几个团队修改同一个系统就紧张。

而我们之所以一直只在小范围做试点，没敢大规模地“推敏捷”，最担心的也是这个问题：如果没有足够的把握，能在这种组织模式下保证系统的质量，即使我们“推”成功了，总有一天还是会退回去的。

那么，怎样保证质量呢？

这个时候就可以翻开敏捷类的书籍了，里面有各种各样的实践：单元测试、持续集成、Code Review……遵循这些实践，我们最终的目标是打造一个质量和风险高度可控的工程体系，并且在这个过程中提升人员的能力，形成团队的文化。

这里分享一下 Google 在工程体系方面的实践。

Google 的所有代码全部放在一个代码库中，除了搜索引擎的核心算法等少数模块，绝大部分代码是没有权限控制的，所有人都可以修改。代码全部是主干开发以及源码依赖，自动化的测试（包括单元测试）、持续集成和 Code Review 是必须的。每次提交代码时，系统会基于最新的源码进行编译，并执行自动化测试。自动化测试不仅会“向下”执行，而且会根据依赖关系“向上”执行所有可能被影响到的系统/模块的 Test Case。持续集成通过之后，还要经过 Code Review，没有问题了代码才能真正入库。

这种做法保证了代码库的基础质量，建立了可以让“任何人随时修改任何代码”的体系能力。这样的工程体系很好很强大，显然要投入相当多的资源和时间去建设，而且需要每一个工程师时时刻刻都用心去维护，这就必须要形成工程质量的文化。

那么，什么是工程质量的文化呢？当我们提到敏捷的团队是面向目标而非系统时，很多主管的第一反应是：“系统没有 Owner，质量怎么办？” 其实潜在的意思就是“是 Owner 的人才会好好干，不是 Owner 的人就会随便造。”这样的文化是不可能把敏捷做好的，系统没有 Owner，就需要每个人都有 Owner 的意识，这才是敏捷的文化！只有建立了这样的文化，敏捷的体系才能够长期有效地运作。

最后分享一个关于文化的案例。

> 我在前一家公司（腾讯）工作的时候，团队也在实施全主干开发、全源码依赖以及持续集成等实践。当时新版的搜索引擎和云计算平台都在开发中，搜索引擎依赖云计算平台。搜索引擎团队有几个开发经理经常投诉云计算平台，因为他们发现很多次持续集成不通过的原因是云计算平台有 Bug、不稳定。他们觉得太影响效率，要求云计算平台发布稳定分支，开发团队基于稳定分支做集成。只有 2 个前 Google 的开发经理说：“没关系，我们就依赖最新的源码，这样才能第一时间帮助他们发现问题，尽快解决，系统就会做得更好。”

大家可以看到，开放的企业文化会造就同样开放和有大局观的员工——不负责 Own 一个系统，工程师们会负责所有。这个案例是公司级的极致的敏捷实践。打造这样的工程体系和文化，成本是巨大的。我们要不要投入这样的成本去换取极致的速度与灵活性呢？或者说什么时候才是“合适”去做这件事的时候呢？我们仍然在探寻中。

2.4 Scrum of Scrums

最近几年敏捷异常火热，很多人在谈敏捷转型、大规模敏捷、困难和应对方法。尤其是大公司，团队耦合严重，牵一发而动全身，想做大规模敏捷转型，不是一朝一夕的事情，基本上都是自下而上与自上而下相结合，部分团队先转型，其他团队跟上，那面对这种情况，我们该如何用敏捷的方式跨团队合作呢？

据不完全统计，Scrum 这几年市场使用率应该是领先的，不仅限于本身流程的实用性，更在于 Scrum 的可扩展性——Scrum of Scrums。Scrum of Scrums 是一种处理多 Scrum 团队相互依赖的复杂机制；是一种团队间协作沟通的方式，目的是保障多个 Scrum 团队顺利并行；是一种紧急程序，当某支团队已经或将要把某些事情放到其他团队身上的时候启用。

随着 Scrum 的应用范围的扩大，复杂的大型项目，跨部门，跨团队，甚至超越传统极限的项目，可以利用管理小团队的管理方式作为基础来进行管理。用管理小型敏捷团队的技巧对于管理多个团队是否有用呢？答案当然是可以的，万变不离其宗，很多处理方式上都是有相似性的，只要合理利用，高效跨团队合作不是梦。在一个大型组织当中，一位 Scrum Master 不仅需要负责各种软件开发和操作团队共同协作，还要负责运维团队、业务员团队等其他团队。Scrum Master 需要清晰地知道各团队日常的工作方式以及工作流程，毕竟改变也是要花时间的，如果 Scrum Master 清晰地知道这些，那么未来各团队在相互合作时，就可以减少不必要的磨合。下面将以携程管家项目为例，来分享整个流程的运转模式。

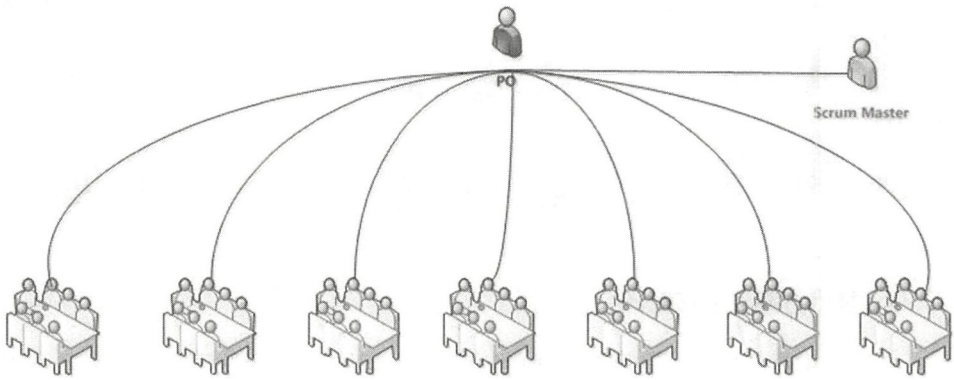
统一沟通方式

在多个 Scrum 团队开发（更多开发者等于更多复杂）的情况下，人太多，必然就会形成一种局面：大家有问题的时候，不知道该找谁，导致自己手头的工作受阻。解决这个问题最好的办法就是建立统一的沟通方式：微信群、邮件组。让大家明确，以后这么多人在一起，就这几种沟通渠道，通过上面的沟通方式，能很快地帮助大家找到对应的人，项目进度、过程问题、风险预警、及时反馈，让所有人知晓。

信息透明共享

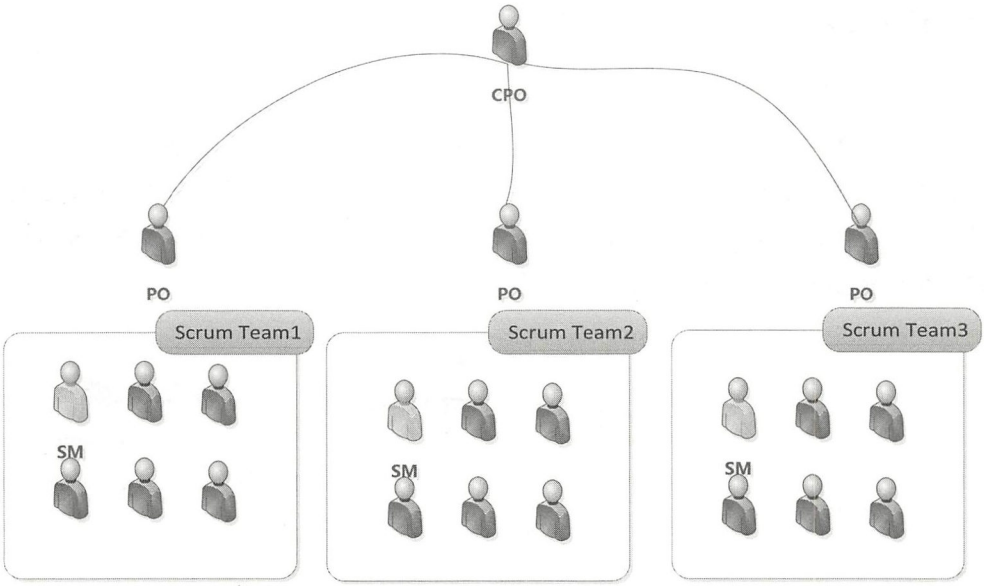
1．干系人

携程管家项目，前后牵扯到开发人员 60 人左右，业务人员若干，涉及团队 15 个，我们很难明确各团队的每个参与人员，但是务必孰知不同团队的重要开发人员，这样在开发过程中，大家遇到问题的时候，就可以自己去找对应团队的人员，进而解决问题，所以重要干系人的清单，是我们需要共享的第一条信息！



2．需求待办清单

在项目初期，我们组织了大范围的需求澄清会，所有的业务方、技术团队都积极参加，目的是希望所有参与者对于需求的理解达成一致，减少后期开发过程中的需求偏差，每支团队都拥有一位产品所有者，他们结合各自团队的能力，使用 Product Backlog 来推动其工作。产品所有者们需要进行合作，这种情况下，往往存在一个“产品所有者领袖”的角色，由产品所有者之一担任，此时，他负责管理横跨各个待办事项列表的聚合视图。



3．技术方案及发布时间

在过程中我们组织了各团队代表的技术方案讨论会，各团队不仅要对于自己团队的技术方案要了解，对于其他团队的方案也需要清楚，在减少技术方案变更的同时，当某团队出现资源短缺时，其他团队能够在最短的时间内顶上。

另外比较重要的就是各个团队之间依赖关系的表格及发布时间，这份信息是需要根据项目实际情况来不断更新的。

各个团队在前期，会对自己团队的开发工作做一个估算，预估发布时间，Scrum Master 需要每周至少一次核实，确保各团队工作正常进行，尽量不要让某团队工作的延误影响其他团队工作的正常开展。

管理活动、问题和风险

不是所有障碍都是一样的，也不是所有在日常出现的问题都需要提交到 Scrum Master 来解决。如果问题和风险会影响到用户故事的交付，或者项目的发布的话，那么就是严重的问题了。问题和风险都会自己暴露出来，可以指定团队中某个具体的人来跟进，及时反馈进度。跨团队以及严重影响整体进度的问题，需要重点关注。

Scrum Master 不需要管理太多细节，他需要相信团队是可以解决和克服这些困难

的，他力所能及地帮助团队解决困难，专注在对项目整体影响最大的点上，那么他就能够更好地避免项目失败。

对风险的了解

Scrum Master 需要对每个团队有足够的了解，并且能够清晰地传达整个方案的整体目标。与此同时，Scrum Master 需要与各团队负责人保持良好的合作关系，让他们提供详细的信息。Scrum Master 不是技术专家，甚者也不是对产品最熟的，但是他可以传达概要信息，让各个团队负责人来传递细节信息。

桥梁

在传统的环境中，直接负责人通常是团队主管，但是这并不是必须的。责任感和驱动力是关键的因素，而不能仅仅考虑公司架构中的角色名字。有些情况下，主管有可能太忙了，或者他明显就不是合适的人选。无论如何，直接负责人是用于具体职责的全新团队角色，可以由团队中胜任的任何人来担任。

在一个实施 Scrum 的组织中，直接负责人应该是 Scrum Master。我认为 Scrum Master 是最佳人选，因为他已经能够掌握敏捷的实践和懂得敏捷的价值。

成功实施 Scrum of Scrums 的关键在于每个和你一起工作的团队都有一个拥护者。对很多方案来说，容易导致失败的关键点是缺乏集中的沟通渠道。通常 Scrum Master 会做大量的工作来启动一个项目，因为这个项目没有直接负责人或者这个直接负责人无法担当起责任，这是一个容易导致失败的关键因素。

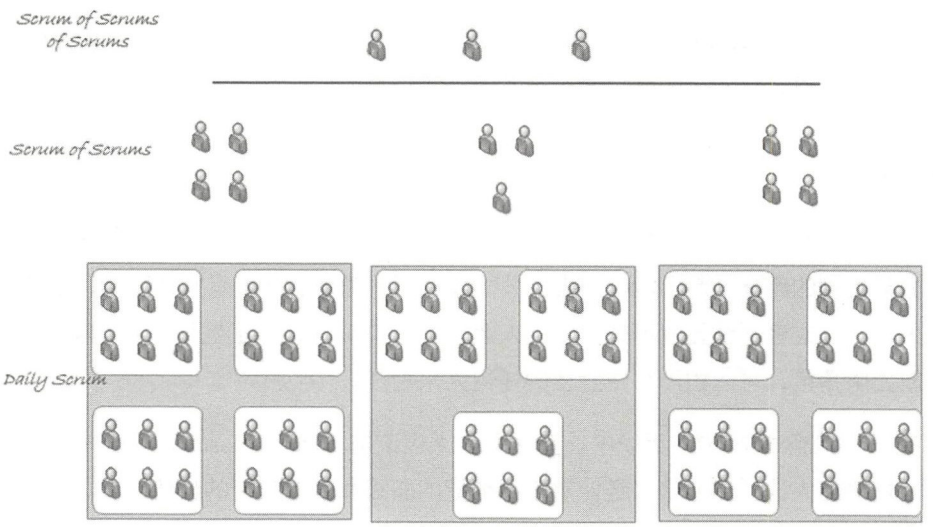
直接负责人并不是指手画脚就可以了。他作为项目和 Scrum Master 的沟通桥梁，必须能够对 APM 的真正意图作出有效明确的传达和解释。直接负责人必须为 Scrum Master 的方案提供数据，同时也需要能够提出问题和发现风险，并且能够和 Scrum Master 一起保证整个方案的顺利进行。

每日站会还是每周例会

这个问题，其实各个团队可以结合自己的情况来定，我们在项目初期，大家相互之间还不是那么熟悉的时候，我们采取的方式是每周 3 次，每周一、三、五，各团队派代表来参加站会。与小团队的每日站会相同，团队代表站在团队的角度，回答以下

3 个问题：我们团队在距离上次开会之间完成了什么？我们团队在距离下次开会之间计划完成什么？我们团队目前遇到哪些问题，需要什么帮助？

当然在项目后期，大家相互之间熟悉了，问题较之前少了一些后，我们就将站会改成 1 周 2 次。慢慢到最后，临近项目尾声，就变成 1 周 1 次了，站会的频率，大家可以根据项目的具体情况，酌情开展。



实施 Scrum of Scrums 是非常简单明了的，而且具有可扩展性，只要你能够掌握原理和基本概念，也明白 Scrum 的价值所在。无论你设计得再好，Scrum Master 都无法百分之百地按照 Scrum 手册来进行操作。为了能够更好地适应大型项目，对 Scrum 的工具进行小幅度的修改是允许的。这样即使以后团队扩张，也能够让团队继续保持敏捷。

2.5 高效跨团队合作，不是梦

同心山成玉，协力土变金。一只蚂蚁来搬米，搬来搬去搬不起，两只蚂蚁来搬米，身体晃来又晃去，三只蚂蚁来搬米，轻轻抬着进洞里。团队合作的力量是无穷尽的，一旦被开发，这个团队将创造出不可思议的奇迹。

携程服务管家项目，致力于利用携程服务优势强化服务的竞争壁垒，提供用户个性、专属、高效的服务体验，建立携程与用户之间的情感线，打造有“温度”的服务，并在此基础上优化员工闲置资源，激发工作主动性，获取更多收入。服务管家项目时间紧、任务重，平台多，1 个月内完成上线，就必须多团队并行开发，如何在如此短的时间内顺利上线呢？答案就是：分工明确、团队合作。

分工明确

团队合作的前提是必须要有明确的分工，各团队都很清楚自己的工作内容，才能在短时间内产生大量价值，提高工作效率，反之如果分工不明确，不清楚要做什么，那造成的后果就是互相推脱，相互争吵。服务管家项目一期主要涉及 4 个平台：携程 APP、独立员工 APP、管家 PC、商户端。项目工期 1 个月，技术团队 7 个，直接参与人数 55 人。各团队分工明确，各自负责各自内容。

团队协作

7 个跨职能型团队，15 个特性小团队，在服务管家项目之前，这些团队与团队之间并没有太多工作交集，默契度也不算太高，所以团队与团队间的合作显得尤为重要，建立沟通机制，同步开发周期，信息透明共享，确定共同目标。

有句话这么说，更多开发者等于更多复杂情况，一个项目并不是参与的开发者越多越好，因为过程中的风险很难把控。面对这种状况，我们首先想到的就是建立沟通机制，保障大家沟通顺畅。如建立专属邮件组、微信群，各团队有变更时，能通知所有人，让大家知晓；明确项目干系人，信息透明共享；每日在微信群中发送日报，汇总今日完成内容，通知明日计划，确保每天进度正常；通知大家目前存在的问题，是希望某些团队遇到难题时，其他团队能给予及时的协助，疏通障碍。

在项目初期，我们组织了大范围的需求澄清会，所有的业务方、技术团队都积极参加，目的是希望所有参与者对于需求的理解达成一致，减少后期开发过程中的需求偏差。另外在过程中我们组织了各团队代表的技术方案讨论会，各团队不仅要对于自己团队的技术方案要了解，对于其他团队的方案也需要清楚，在减少技术方案变更，当某团队出现资源短缺时，其他团队能够在最短的时间内顶上。

鉴于酒店无线团队之前在敏捷开发上运行得比较有成效，这次我们同样使用了

Scrum of Scrums 的方式来运转，每周一、三、五，各团队派代表来参加 Scrum of Scrums 的站会。面对面的沟通，不仅加强了团队间的合作，增强信任，也让各种难题顺利解决。

55 个人，来自不同团队，虽然大家职责不同，却有着共同的目标：1 个月，项目准时保质上线！这是一个大的目标，拆分到 4 周，1 周 1 个 Sprint，1 周 1 个中等目标；细化到每天，每天 1 个小目标。面对如此频繁的联调，我们使用了 War Room，让大家坐在一起，遇到问题，不再发送邮件，不再打电话，不再 Lync，直接到有问题的地方去，目的就是为了今日目标的达成，各团队相互合作，彼此达成默契，哪里有问题，就及时奔赴“战场”！

团队内部协作

团队间友好合作的基础是多个坚实的团队，而我认为要形成一个坚实的团队，最重要的就是团队精神：相互信任，相互帮助，相互包容，相互谦让。小溪只能泛起破碎的浪花，百川纳海才能激发惊涛骇浪，个人与团队的关系就如小溪和大海。每个人都要将自己融入集体，才能充分发挥个人的作用。团队精神对任何一个组织来讲都是不可缺少的精髓，否则就如同一盘散沙，一根筷子容易弯，十根筷子折不断。本次参与携程服务管家项目的 7 个团队都是由跨职能型团队组成的，每个团队的团队精神都得到很好体现。

相互信任，团队内部的每个成员都应该相信其他团队成员，总是抱着怀疑的态度与其他成员沟通，不信任团队就有可能造成很大的损失和误会。本次服务管家项目的开发工作，都是并行开发，对于每个需求，开发人员向测试人员都承诺了提测时间，以便给予测试充分的时间来保障测试质量，并且测试人员也可以提前做一些准备工作，比如说测试数据、测试环境等。如果开发人员延期了，在规定的时间内没有提测，那么测试人员的提前准备可能就会被浪费掉，那随之而来的就是抱怨，互相争吵，然后恶性循环。当然正是因为团队内部有了这种信任，才能让大家合理利用自己的时间，更好地去合作，避免一些不必要的浪费。

相互包容，管家作为一个创新型项目，市场没有太多相似内容可借鉴，我们也是抱着试一试的态度，去看看用户的反馈，那么带来的影响就是需求的变更，我们很难保证产品经理的需求不变更，市场在变化，如果团队内部大家一味地去指责产品经理，

那肯定玩不下去，所以只有包容才能和谐。一个团队共事，难免有发生矛盾和误会的时候，这时就需要我们有一种相互体谅，一种相互包容的博大胸怀，非原则的问题不要斤斤计较，较真不仅影响工作，影响感情，也影响团队凝聚力，是没有任何意义和价值的。

相互帮助，说起容易做起难，要想做到相互帮助，就要求团队每个成员都要树立全局观，不能只顾自己的利益，要将个人利益融入团队总体目标利益中去。比如这次的酒店无线前端团队，团队内部有个服务端同事犯胃病，住院十天，作为服务端主力开发人员，这个消息无疑对团队的打击很大，所以我们很快召集前端团队开会，让大家一起想办法看如何解决，最后几个客户端同事自己站出来，提出参与服务端的开发，减缓这个“小意外”的影响。试想如果这些站出来的客户端同事不同意帮忙分担服务端的开发工作，甚至他们觉得自己的工作凭什么无缘无故加大了这么多，而且又不是自己的分内事，那么可想而知，本次的目标肯定是无法完成的，所以要想完成目标，就必须学会团队成员之间相互帮助，以大局为重。

相互谦让，一个团队中难免存在竞争关系，在如今公司体制下，大家都想升职加薪，都想去表现，做多做少都会去争去抢。或许很多人认为只有傻子才会去谦让，其实在我看来，谦让是相互的，要形成一个有凝聚力的团队就必须做到这点，不然每个人都斤斤计较，那么这就不是一个团队了。

一个团队的每个人，能力素质可能不同，岗位角色可能不同，生活方式可能不同，但是只要挑战极限的勇气相同，勇创一流的志向相同，捍卫团队的行动相同，这个团队就一定能创造出一片天地，这就是团队精神重要性的直观表现，也是我理解的团队精神，也是团队精神的重要所在。

明确的分工，完美的团队间及团队内合作，最终让这 55 个人在 1 个月内保障了服务管家一期的顺利上线！协力致胜，团队为赢！

第 3 章

打造幸福的小团队

敏捷的核心在于团队的成长，在打造高效小团队的同时，幸福指数的提升也是我们的团队目标之一。

3.1 幸福满满的团队成员

经历了从瀑布模式到敏捷开发模式的研发人员，从刚毕业的应届菜鸟到产品老兵，从 360 考评到“十分感谢”，转型过程在造就团队凝聚力的同时，也带来了团队成员幸福感的提升。

3.1.1 从团队成员的视角看敏捷

接触敏捷是加入 Scrum 团队的时候，在此之前，我们一直处于瀑布型的工作模式。我的 Scrum Master 曾经问过我一个问题：你觉得敏捷和瀑布比较有什么区别吗？你觉得哪个更好？我无法从好坏的角度去评论，每个过程都有它独特的优势，然而经历一年的敏捷实践确实给了我不一样的体验。

我是一名测试工程师，瀑布模式的时候，在我拿到入测的可交付成果之前，我负责的是了解契约，了解需求逻辑，编写测试用例，准备测试数据等一系列的准备工作，入测以后就是执行以及验证的工作，所有的工作都是按照计划进行的。存在的最大问

题是开发人员每次都要在完整做完需求后再提交测试，缺陷的爆发期非常集中，以至于项目在上线交付过程中“带伤”上线的概率较高，更有可能出现残次品。

然而进入敏捷模式以后，对于我的第一个挑战是估点。需求澄清会是加入 Scrum 团队后参加的第一个会议，会议时长 2 小时，在一个小会议室里，或坐或站地挤满了人。

1．需求讲解：PM 们根据表格上的顺序依次讲解需求的说明，要做什么，这个需求的背景是什么，我们为什么要做它，做完它会带来怎样的收益（当时还不知道那个有顺序的表格就是 PB Backlog）。

2．需求答疑：PM 讲完后研发人员针对需求不明确的内容提出疑问，然后答疑，直至需求明确。

3．需求补充：开发就一些需求从技术实现上提出更好的方案，测试根据自身的经验帮忙补全一些 PM 未考虑的异常、边界场景。

4．需求估点：大家拿出扑克牌给出对于该任务的估点（最懵的一个环节，也是后面对我启发最大的一个环节）。

5．需求取消：有一位 PM 的需求收益数据未能真正说服团队，该需求在一番讨论后被取消了排期。经过 2 小时的会议，作为一名新人的我虽然无法完全明白所有的业务逻辑，但是接下来两个星期的规划在我心中有了一个清晰的认识。

第一个 Sprint 周期后开始正式进入项目，因为自身的经验问题以及团队的状态问题，导致了逻辑修改后开发偏高的缺陷率，时不时环境挂掉半天等一系列的意外，这些打得我措手不及，加班变成了一种常态。Scrum 的快迭代难道也是靠加班赶工吗？

其实不然，在后续一年的 Scrum 实践过程中，我渐渐悟出 Scrum 和加班并没有关系。Scrum 是用一种可持续的稳定的节奏来降低以往频繁出现的最后一秒临时救火的不可控场面。它是以经验主义作为理论基础的过程。经验主义主张知识源于经验，以及基于已知的东西做决定。所以经验越丰富的团队的成熟度和对于承诺的把控度也就越高。而经验的丰富源自于不断地学习和改进实践。

经过一年的 Scrum 实践经验，我的经验得到了快速提升，不仅习惯了敏捷的每日站会、需求澄清会、计划会议、Demo 会议、回顾会议，更养成了一种持续学习持续改进的习惯。团队间也把精力不仅仅放在关注进度、计划细节上，并且更多地放在如

何相互合作、产出最高价值的产物、共同的目标与共同进步上面。

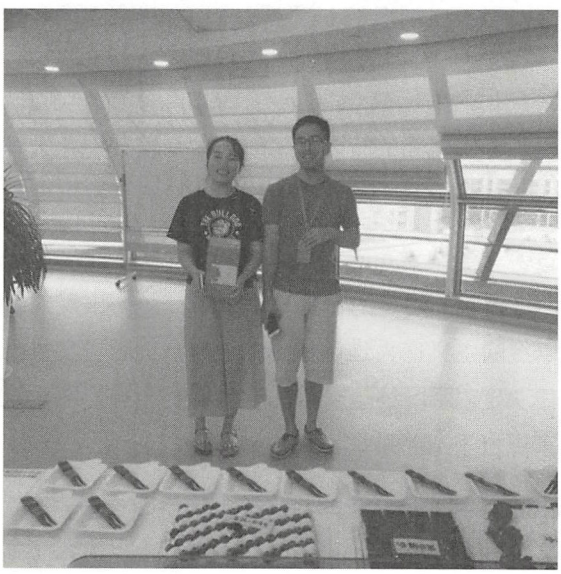
3.1.2 与团队共成长——产品新兵成长记

2014 年 11 月，经过多轮的校招面试，我以技术培训生的身份加入了携程机票无线事业部实习，成为机票前端的产品经理。光阴如白驹过隙，转眼三年过去了，曾经的产品新兵成长为一名能独立负责产品业务的 PM，同时也见证了很多跟自己一样初来乍到的新人如何在团队的帮助下快速成长。

实习导师引领，快速进入角色

作为刚刚步入职场的新人，内心大都充满了好奇与期待，同时也满是不安与忐忑，陌生的工作环境，陌生的工作伙伴，想快速上手工作内容，进入角色，似乎有点无从下手。

对于新人的顾虑，部门似乎早就摸得一清二楚，新人尚未报道，一位与之缘分匪浅的实习导师就已经安排好了。实习导师会从各方面帮助新人上手，大到公司和本事业部的战略与规划，小到餐厅怎么走，事无巨细，如春风化雨般消解新人初来乍到时的不安。导师给新人送上生日礼物，准备了生日蛋糕。



还记得，为了让我尽快了解产品经理的职业角色，学习 PM 的基础专业技能，我的实习导师先是找来了过去近两年无线机票产品迭代的需求文档，让我了解产品的迭代历程，然后从简单易操作的需求入手，教我如何把控需求，如何与交互、视觉和研发团队沟通与协作，将需求落地，并跟踪上线后的数据。导师的倾囊相授，让非技术背景、非专业出身的我，逐渐进入了产品经理的角色，找到了做 PM 的感觉。

在机票无线团队，每个刚入职的新人都会有一个导师。导师不仅帮助新人快速上手工作，融入团队，更重要的是会让新人在团队中能找到一种归属感。当新人成长为业务骨干，有一天成为另一名新人的导师时，那些曾经他所受到的关照、学到的知识，也悉数教了出去，这大概是对自己的导师最好的回报。

团队帮扶，成长进入快车道

团队中的每个人大都各有所长：有人精通业务，对机票领域的业务知识信手拈来；有人擅长数据分析，各种数据库和字段烂熟于胸，取数和分析数据的能力不比专业的分析师逊色；有人逻辑缜密，产品需求文档写得滴水不漏。

在机票无线团队，分享和“老带新”的氛围非常浓厚。各有看家本领的团队伙伴，在有新成员加入的时候，只要能帮一把，从来不吝惜时间和精力，尽管自己忙得脚不沾地。

团队有一块专用的上课小白板，每次团队中有新人请教老成员某一方面的机票知识时，老成员们要么在座位临时解惑，要么就是吼一嗓子，“来，要讲机票的 XXX 内容，要听的都过来”，于是三五个新人围过来，边听边提问，边学边记笔记。白板小课堂，短的十几分钟，长的半个多小时，一拨又一拨的新人就这样在这块白板前被领进了机票业务的大门。

除了小团队内的学习和交流，新人成长也少不了在很多大团队内的学习。

还记得，为了让我快速地熟悉和了解机票业务，精通业务的同事不仅自己多次讲解，还帮忙联系让我到机票预订部的各个小组学习，了解整个机票的预订前和预订后的流程。

刚开始实习时，作为一个不懂半点技术的产品新人，与研发同事的交流常常让人感受到是一种交流的无奈，明明大家都说的是汉语，但是彼此却听不懂对方在说什么，

可偏偏我的问题还很多。常常，我屁颠屁颠地跑到开发同事的座位，珠连炮式地一连问了好几个问题，开发同事噎了半天，然后以一两句很固定的句式开头回答我“我没听明白你说什么，你能再说一遍吗？”后来我才知道，开发同事写代码兴致正浓，一个产品新人过去打断，问了一堆不知所云或简单到低级的问题，是多么招人烦，然而当时却没有一个人嫌弃我、敷衍我。

正是团队中每个人给予的帮助、耐心和包容，新人才得以放开手脚大胆做事，进入成长的快车道。

在大项目中试炼，培养团队默契

在机票无线事业部，刚毕业的新人在经过一年至一年半左右的训练与成长后，会有机会独立负责大型项目。这是每个人都渴望已久的时刻，就好像上山闭门勤学苦练了十年，现在师傅终于给你下山的机会了。

在毕业一年多的时候，我也迎来了这个机会。2016 年 10 月，机票 H5 启动了 Swift 国内主流程改版项目，旨在通过这次改版，实现机票 H5 转化和收益的双向提升。在项目正式启动之前的调研期，我兴奋得就像打了鸡血，我渴望通过这个项目试炼自己，检验自己掌握的各方面产品技能，同时与团队进行更深入的合作。

在近 5 个月的项目周期内，我们遇到了很多难题。浏览器进入方式太多导致 AB 实验分流串版，实验数据不准；改版后的单页应用如何兼容 SEO 投放；使用新站点新 url 之后如何把老版本的流量逐步切换，并且不对 SEO 的排名造成负面影响……在这个过程中，我们会彼此抱怨、互相吐槽，发泄心中的愤懑，也会彼此安慰，互相鼓励，一起坚持。当改版完成，机票 H5 的订单转化和收益终于实现双双稳步提升时，团队中的每个人都激动到哽咽。

机票 H5 Swift 国内主流程改版项目，项目组包括产品、开发、测试、交互、视觉、用研、BI 等多个团队的成员，有不少人都是最初一起入职的新人。经过这个项目的试炼，不仅项目中的个人取得了长足的进步，团队成员之间也更加信任和默契，整个团队的互助、包容和协作也达到了一个更高的水平。

在团队转型中共同进步

2016 年，机票 H5 开始往敏捷开发团队转型，以进一步提高团队的工作效率。作为机票前端第一个敏捷开发团队，对于转型我们一开始是抱有怀疑的。在过去近两年的时间里，团队成员之间已经形成了固定的合作模式，一周迭代一个版本上线，基本已经是人尽其用，走 Scrum 还能多大程度地提升效率呢？

带着质疑，我们组建了机票第一个 Scrum 团队，开始学习敏捷团队的运作模式。需求评审会全员参与估点，需求计划会快速抛出问题、确定功能提测时间，早间站会每人更新任务进度，每月的项目回顾会一起制订下个阶段的项目运作优化目标。对于 PM 来说，全员参加需求评审会，对需求的理解会更透彻，在开发和测试过程中，遇到的问题也相应减少，但感触最深的却是每隔两周或每个月的项目回顾会。

在项目回顾会上，产品、开发、测试人员都会分别来说一说在过去的项目周期取得了什么样的进步，有哪些不足，未来计划如何改进。不论是前阶段的项目进展顺利，或是遇到较多的阻碍，在项目回顾会上，大家都会很虚心地来做自我检讨，反思自身的不足，并对自己提出改进的要求。有时大家也会互相吐吐槽，产品的 PRD 提交晚啦，开发的估时不准确啦，互相提要求和改进点。通过项目回顾会，我们认可自己的进步，发现自己的不足，并共同制订下一阶段的团队优化目标。作为团队中的 PM 来说，感受最深的是，我们更加理解彼此，信任彼此，团队更融洽更友爱，产品迭代效率提高成了一件自然而然之事。

经过三年的磨练，我从一个刚毕业的校招生，成为了一名相对成熟的产品经理，除了自身的努力，更离不开团队的帮助。三年来，每到毕业季，就有许多跟我曾经一样的新人加入携程机票团队，并在团队的帮扶下一步步成长起来，成为新一代的携程人。

3.1.3　“十分感谢”——让大家都成为“小太阳”

在我们的团队中，经常会有一些“孙悟空”式的员工，他们聪明、积极、专业能力强、办事效率高，总是能把工作做得很出色，但是在团队合作上很让人挠头：做事情“挑活儿”，只做大的、重要的工作，小的、不出彩的丢给别人；对待同事经常“看不起”，觉得别人这不行那不行，总是批评和挑剔，经常是工作完成了，但是不落好儿。也有一些“沙和尚”式的员工，能力可能并不那么突出，但总是勤勤恳恳地工作，

默默地承担别人不愿意做的工作，认真地帮助他人，是大家都愿意合作的对象，但是在以结果为导向的绩效考核体系下，这种人通常会比较吃亏。

作为管理者，我们希望能够可视化每一个成员对团队的贡献。让“孙悟空”们意识到自己的问题，在自己努力向前冲的时候也能够提携和帮助别人，从而在团队中发挥更大的，并且是正面的影响力。同时，“沙和尚”们的贡献也需要让大家看到，并且要给予奖励，从而给团队树立正面的榜样，鼓励大家多为团队做贡献，在成就团队的前提下成就自己。

那么，怎样让大家清楚地看到每一个人对团队的贡献呢？常用的方法有两类：一类是人力资源管理中常用的“360 度考评”，一般是让 3~5 个相关的同事按照一定的标准给考评对象打分、写评语，比较多用于管理者晋升评定等场景。另一类是业务/技术团队中常用的“积分制”，就是设定一些条件，比如当一个人做了诸如分享、带新人、组织团队活动等事情时，会给予一定的积分。定期对积分排名靠前的员工进行奖励，或者要求累积到一定分数时才有资格晋级或者得到 A 级考评等。

但是当我们试图采用这些方法解决上面的问题时，我们遇到了问题。360 度考评最大的问题是评分受主观性因素影响太大，打分者本人的严格程度、打分者与被评者关系的亲疏程度，对最终的评分结果影响很大。这就导致了 360 度考评只能是一种“背靠背”的方法，只能用来对个人进行评定，而不能在团队成员之间进行比较。而积分制最大的问题在于它是对行为的评价，而非对结果的评价，所以容易导致“刷分”行为。曾经有团队，在发布了积分制之后，每到晋升或考核季之前，就有一大批的业务/技术分享活动，我们在课后做抽样调研“你觉得讲得怎么样？”“还行吧。”“对你的工作有帮助吗？”“有一点吧。”——大家可以想象，这种抱有功利目的、临时抱佛脚的“团队贡献”，会比较容易流于形式，拿这个积分来竖立团队的榜样，很多人是不服气的。

于是，我们开始思考，试图找到一个方法，能够可视化每一个成员对团队的贡献，同时它还必须是公平的、客观的、结果性的、正面导向的、不会“通货膨胀”，以及简单、可操作的。最终我们找到了这个方法，并给它取名“十分感谢”。我们希望所有人，对于曾经帮助过自己、帮助过团队的人，都怀有感恩之情，并以同样的态度和行为去对待他人。“赠人玫瑰，手有余香”，如果每一个人都努力成为团队的“小太阳”，这样的团队一定是团结的、强大的。

“十分感谢”的操作过程十分简单，分成 3 个阶段：

1．第一阶段：评分。每位同事有 10 分，可以送给团队内任何他想要感谢的人，可以都给一个人，也可以分给不同的几个人，只要说明感谢的理由就行。

2．第二阶段：汇总得分。汇总每位同事收到的分数，得到排名。

3．第三阶段：反馈。

- 每位同事可以分别看到自己收获感谢的分数以及原因。
- 得分最高的几位同事会在季度会/半年会/年会上进行公开表彰。
- 得分较低的同事，领导一般会进行单独谈话。

下面是一个 50 人左右的团队，年度“十分感谢”的具体操作如下：

第一步：邮件通知与规则说明

XX 部门“十分感谢”大奖评选

评选流程：

1、 每位同学有 10 分，可以送给任意你想要感谢的人。

2、 请在下周二之前完成，地址是：https://www.xxxxxx.com/x/xxxxxx/。

3、 我们会分别统计每位同学获得的感谢分，得分最高的几位同学将会获得年度“十分感谢”大奖。

4、 会议结束后，每位同学可以分别看到自己收获感谢的原因。

几点说明：

1、 送分的单位为 1，不可以送 0.5 这样的分数。

2、 送分总数不超过 10 分，送给谁都可以——可以把 10 分全部送给一个人，也可以分别送给几个人。每人 N 分，说明你感谢他/她的理由就可以了。

3、 这次的送分范围是 XX 部门内所有同学。鼓励跨组送分，只要你认为他在过去一年对你有帮助。

4、 不能送分给自己。

5、 为了便于统计，所有姓名填写中文名。

6、 希望大家尽量把 10 分都送出去，但是送不完 10 分也没关系。

7、 5 个经理不参与被送分，当然我们会保留感谢别人的机会。

8、 所有的送分记录以及得分统计的信息是保密的，除了 5 个经理外不会有其他任何人知晓。

第二步：评分

每位同事填写一份这样的表格：

送分人	送分给谁	分　数	感谢的话
小丽	玉姐	4	感谢玉姐的指导和监督，让我学到了很多，也让我的工作越来越得心应手，玉姐是我学习的榜样
小丽	旋哥	2	旋哥在很多地方给我这个新手帮助，即使在很忙的时候也抽时间，大赞
小丽	冰冰	2	我俩都属于新同事，互相学习，互相帮助
小丽	文文	2	感谢文文替我们组织了那么多的活动，新时代的活雷锋

送分人	送分给谁	分　数	感谢的话
小强	天哥	3	经常分享各种技术，太有用了
小强	倩倩	3	没有倩倩在 XX 项目上的努力，这个项目做不成今天的样子，榜样
小强	玉姐	2	感谢各种忍受我的骚扰和问问题
小强	芳芳	1	开发的 XX 工具，对我们的工作很有帮助
小强	冰冰	1	感谢每天让我蹭车。

第三步：汇总整理

组织者汇总整理后：

得分人	送分人	分　数	感谢的话
玉姐	芳芳	5	感谢这一年对我工作的支持和帮助，感谢真诚和热心
玉姐	小丽	4	感谢玉姐的指导和监督，让我学到了很多，也让我的工作越来越得心应手，玉姐是我学习的榜样
玉姐	志哥	4	从年初的数据库方案开始一直在解决各种问题的第一线，辛苦了
玉姐	小强	2	感谢各种忍受我的骚扰和问问题
玉姐	潇潇	2	在玉姐的支持下确定了 XX 项目的对接方案，使系统从创建到使用有了闭环
玉姐	……	……	……
总分		35	

得分人	送分人	分　数	感谢的话
文文	思思	3	与部门其他同事相比，跟着你工作的时间最长了，教会我从一个学生转变成一个职场员工。教会我做事要踏实细心
文文	玉姐	2	感谢文文这一年勤勤恳恳的付出，默默地做出了很多成绩，XX 系统设计得非常棒
文文	小丽	2	感谢文文替我们组织了那么多的活动，新时代的活雷锋
文文	倩倩	1	常常帮助大家组织活动，任劳任怨，是团队里面的“老黄牛”
文文	……	……	……
总分		15	

第四步：总结分析

团队的主管们在一起检查了所有的得分情况。

对于成员对团队的贡献，主管们心中其实是有大致判断的。所以我们首先看总体排名情况，尤其是排名靠前和靠后的同事，是不是跟平时我们观察到的、感受到的是一致的？结论是，相当一致（事实上，“十分感谢”项目在很多团队做过，结果做出来与主管的判断基本都是一致的，说明这种方法的有效性很高）。

之后我们看得分的总体分布情况。按照规则，假如每人都送完了 10 分，应该平均每个人也得到 10 分，所以 10 分是个中等表现，高于 10 分是比较好的，低于 10 分就是比较差的。对于 50 人左右的团队，能够得到 20 分及以上的同事，我们认为是比较优秀的，而 5 分以下的同事，就需要注意了（刚入职，仍处于学习阶段的新人可除外）。

确认有效性之后，我们进一步分析大家送分的理由，最后归结为 3 类：

1．对他人的工作/成长有帮助。这是送分最多的理由，大约占了全部送分的 70%。

2．工作能力或者工作态度表现出色。被视为榜样或者好战友，大约占 20%。

3．工作之外的各种服务/支持。比如组织活动、订餐、搭车、带来欢笑等，大约占 10%。

这个过程中还发现了一些有意思的现象：

1．有些“孙悟空”的得分并不低。虽然在工作中经常有人抱怨他态度不好，但是会有一部分人觉得他做事靠谱，在合作过程中能够学到东西，仍然表示了感谢或者崇拜。所以对于“孙悟空”类的同事，是否一定要让他做出改变，可以酌情考虑。如果得分尚可，说明对团队的正面影响多于负面影响；如果得分太低，则要考虑谈话了。

2．默默奉献的“沙和尚”同事，如果在专业能力上没有突出之处，只能在工作之外服务和支持大家的话，通常会得到一个中等略偏上的分数。这一方面说明他对团队的影响是正面的，同时也反映出对团队的带动作用是有限的。

所以最终的结论是：在“十分感谢”中得到高分的同事，都得是德艺双馨的啊！

第五步：反馈

为了保留惊喜，我们没有直接发布结果，而是先在部门年会上举行了一个既隆重又温馨的颁奖仪式。台上领奖的同事们很开心，因为这是团队对他们付出的肯定。台下鼓掌的同事们也很开心，因为走上领奖台的是大多数同事都喜欢的人。

开心的事情大家都愿意做，然而，不开心的事情，却也不得不做，那就是与得分低的同事进行谈话。这个时候，“十分感谢”这个机制的有效性就显示出来了：以往我们跟员工做这类谈话，往往依据的是其他人的投诉/抱怨，或者是抽样的 360 反馈。而被谈的同事经常是不太服气的，他们会认为别人嫉妒他，或者对他有意见，故意诋毁他。但“十分感谢”是一个全员参与的活动，得分低意味着“没有人喜欢你”。3、5 个人不喜欢你，可能是他们的问题，但一个团队几十个人，如果没有人喜欢你，那一定是你自己的问题。所以被谈的同事在看到自己分数的时候，通常自己就会意识到有问题存在，这样谈话就会比较容易进行了。至于为什么得分低，他能不能够改进，怎样改进，这就需要主管们根据情况自己去判断和去寻找答案了。

“十分感谢”不仅能够用在团队内部，也可以用在合作团队之间“互评”。比如我们也做过让产品和技术团队互相送分，然后分别做出得分的排名，同样可以看出来谁在合作过程中发挥了更好的作用。

总结一下，“十分感谢”是一个比较客观和有效的评分机制，能够反映出每一个团队成员对团队的贡献。在这个基础上进行适当的反馈和引导，鼓励每个成员都更多

地帮助他人，成为团队中的“小太阳”，那么这个团队会更有可能成为一个团结的、强大的团队！

3.2　幸福指数爆表的团队会议

从游戏中总结工作成果的回顾会，喝着碳酸饮料的“碳酸分享会”，全新模式的团队会议给团队注入不一样的幸福活力。

3.2.1　不一样的回顾会——让团队都有所收获

回顾会对于敏捷团队来说是个非常重要的会议，它能够帮助大家总结在这一阶段的好与不好，目的在于让团队都能从中有所收获。敏捷转型过程中，回顾会也在转型，以下是我罗列的在敏捷转型过程中根据团队的不同阶段调整的几次不同模式的回顾会案例。

“good，could have been better，important”回顾会

1．团队现状

H5 团队是机票部门第一个尝试敏捷转型的团队，之前都是小版本发布+每周常规发布的模式，大家普遍觉得节奏很乱。在这种模式下，大家的精力会被分散，无法专心做事，容易被打扰，很多同事都在多线程操作。

例如，做 V2.6.0 计划月底上线的需求开发任务时，还要继续跟进每周的常规发布任务，并且都是一些产品临时加的变更或是线上的缺陷，有些任务还特别紧急，甚至会影响原本月底上线的 V2.6.0 的项目计划，这无疑给研发同事们带来了巨大的压力，除身心疲惫外，整体感受也非常凌乱。产品经理也很无奈，有些是政策任务还都不得推脱，有些是上个项目上线后数据不好看，方案得重新调整，这些变更都变成了临时的常规发布任务。

2．尝试转型

开发经理提出想尝试调整这种发布模式，我们既然每周都有发布，我们为何不能

将版本发布和常规发布的模式合并，我们可以这周决定下周要做的事情，大家也可以更加专心在一件事情上。H5 团队本身与 APP 团队不一样，H5 不用跟随 Native 版本，我们本来就有随改随发的优势，我们的开发测试团队一直都是固定的一批人马，我们不就是一只现成的敏捷团队吗？于是我们立马调整了的节奏，我们每周只做上个计划会上按照优先级排好的事情，人员分工都清晰透明，在排计划的时候同时也预留了一些小 buffer 来应对接下来一周中可能出现的小变更。

3．总结回顾

这种模式运行了 1 个月后，SM 组织开展了一次以“good，could have been better，important”为主题的回顾会，让大家回顾在模式调整后，大家做得好的以及一些需要改变和改进的地方。

回顾会上大家都非常积极，在回顾大家这一段时间以来觉得做得好的时候，大家一致觉得能更专注地做每周计划的需求了，没有临时的常规需求打扰，不会被分心，需求上线速度也快了，当然需求上线速度快这也是产品经理最开心的地方。现在调整为每周一次发布，我们之前穿插的紧急发布也少了。之前要版本+常规两套分支，现在只管每周一次的发布分支，这样维护的分支也少了。每个人的任务都透明化了，效率也提升了。SM 把大家讲的所有好的一条条记录在便签上，贴在白板上。所有人都信心满满，这也是整个会议过程中大家最激动的时刻，大家都洋溢着满意的笑容，觉得我们的团队简直太棒了！

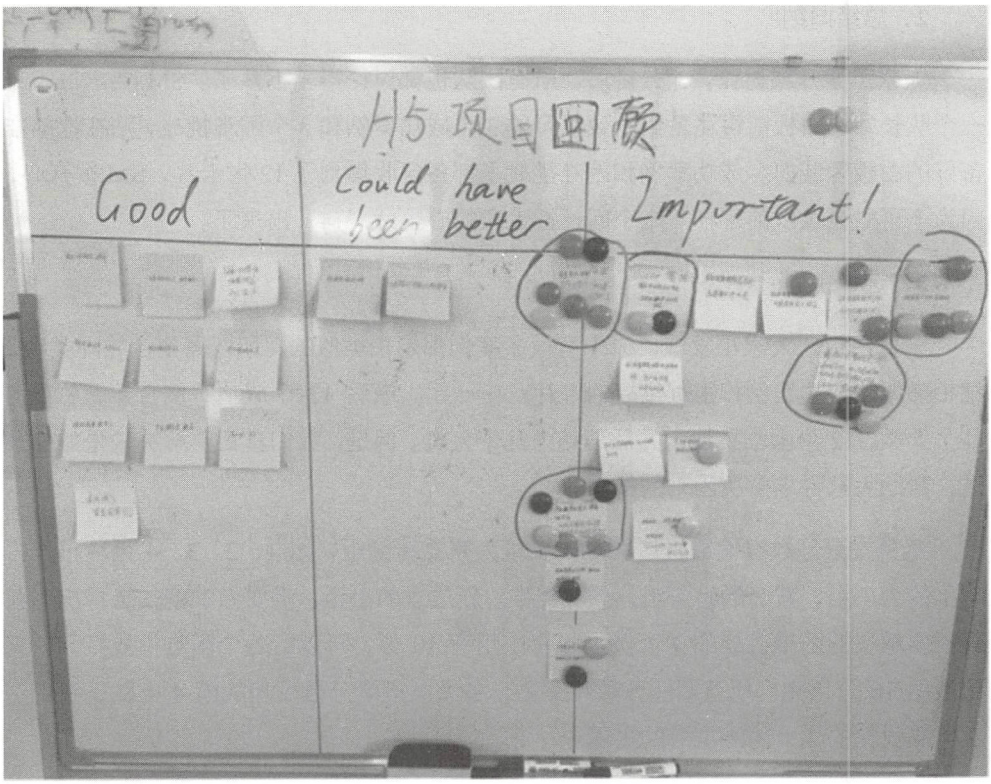
4．面对困难

以前，我们都害怕面对自己的错误，或者知道有问题，也没有好的解决方案，一说到“反思”和“检讨”，大家都很排斥。但这次不一样，我们开始了一段特别激情的“good”回顾铺垫后，大家开始要面对问题的时候表现得特别坦然，甚至有些期待，因为大家希望团队能变得更棒，觉得有再多的困难，大家都能面对。SM 也换了种问法：“大家想一想，我们这段时间做得这么棒，如果重来一次的话，大家哪些地方调整一下会做得更棒呢？”

所有人都开始沉默了，几分钟后，几个负责人首先发言：我们产品前期调研还是不够，我们的契约没有发起评审，很多干系人都不清楚。大家的主动意识还是弱了一些，此刻越来越多的声音涌现出来，SM 把大家的问题一一记录下来，用另一种颜色的便签贴在白板上。

5．棘手问题

原来我们还有很多需要改进的地方，但是这些问题是否都是最高优先级需要解决的吗？SM 发给大家一些磁铁，让所有人给这些问题投票，决出当前最需要解决的 TOP 5 问题，同时每个问题由团队决定分配一个对应的 Owner 去跟进这些问题。最后大家一致决定，下一次的回顾会还是放在一个月后，同时需要检查这次回顾会的 TOP 5 问题的落实情况，是否都得到妥善解决。



“游戏互动”回顾会

开了几次以“good，could have been better，important”为主题的回顾会后，大家开始不知道说些什么了，发言的人越来越少。SM 自己也感觉到每次都是这些问题，于是决定以游戏互动的形式开一次全新模式的回顾会，围绕着大家平时在敏捷推进、项目开展过程中碰到的一些问题去寻求答案。

1．分组讨论

在约定好的时间，大家都准时来到会议室。SM 清点了到场的人数，16 人，盘算正好可以分成 4 个小组，于是让大家快速组队，SM 给每组发了 6 张卡片，其中 3 张让大家用一句话描述最近一个阶段大家一致赞同的成功案例，另外 3 张卡片也用不同颜色区分，让大家用一句话描述最近一个阶段一致觉得还需要继续挑战的地方，要求字体要大，只需一句话即可，讨论 3 分钟。

2．总结归纳

大家迅速开展讨论，3 分钟后，所有团队也都讨论得差不多了。SM 要求每组派一个队长站在白板前讲述各组一致认可的 3 个成功案例和 3 个困难挑战，贴在提前准备好的白板区域内。成功案例和困难挑战里都各自收集到了 12 张卡片，SM 很开心，因为不再需要像以前那样一个个问，把大家的心里话“抖”出来了。

3．选择主题

一览白板，大家都发现，原来有很多案例都是相类似的，所以需要合并同类项，把相类似的成功案例和困难挑战的卡片归在一起。最后 12 个成功案例合并后总结为几个大类，12 个困难挑战合并后总结为几个大类。但是，我们真正要解决的是如何改进，帮助大家寻求解决困难的答案。

避免投票分数有偏重，这次 SM 要求大家重新分组，以 1、2、3、4 循环报数，喊 1 的为一组，喊 2 的为一组，依次类推。新成立的小组，需要做的就是选择出当前最需要解决的困难挑战的 4 大类。每个小组有 10 分，可以给各个困难挑战打分，直到最后 10 分用完。所有团队分数用完后，各大类投票分数相加的前 4 类即是我们这次需要讨论的 4 大困难挑战的主题。

4．想法和建议

每个组认领一张困难挑战，贴在原先已准备好的白纸上，作为他们当前讨论的主题——如何给出帮助来解决这些困难。队长负责收集归纳大家的意见，写在白纸上。

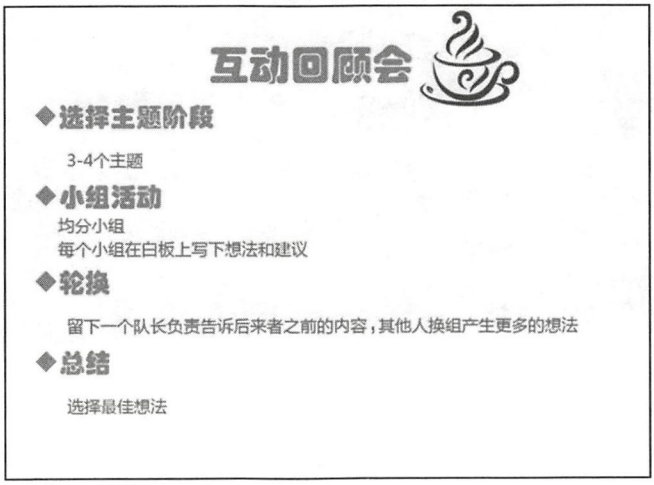
5．轮换

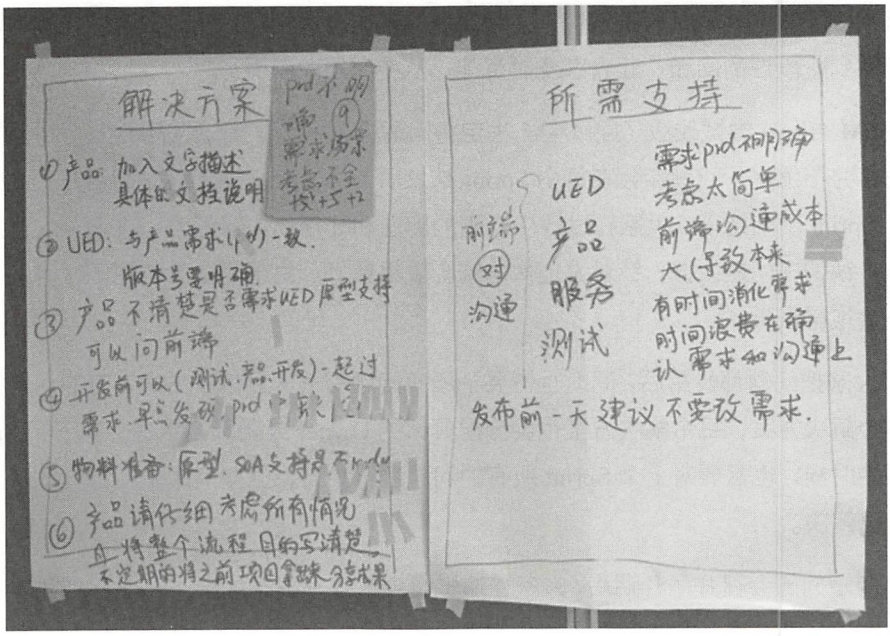
讨论 3 分钟后，其他的组员按顺时针轮换到下一组，帮助另一个组解决困难，提出更多想法和建议。留下的队长负责告诉后来者之前讨论的内容，直至所有组都已轮

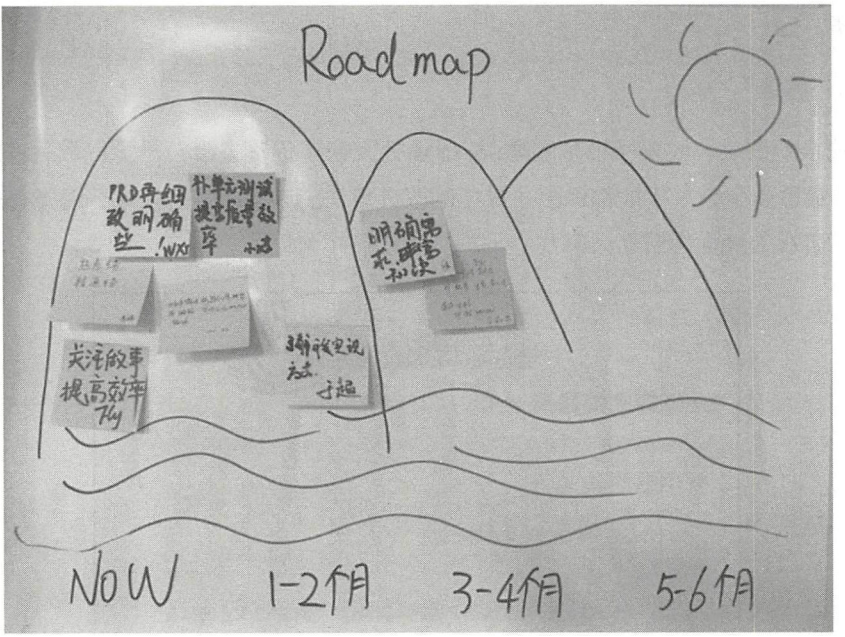
换完毕。

6．总结

轮换一圈后，收集了很多的建议，SM 让大家用标签去给自己觉得最满意的建议投票，最后 4 个困难挑战都诞生了最佳的改进想法。这次回顾会大家普遍觉得非常有趣，在游戏的过程中帮助大家找到困难并寻求到了帮助。







“简短”回顾会

游戏互动固然有趣，但一次下来快则 1 小时，慢则 1.5 到 2 小时，非常花费时间。大家平时工作中时间都非常紧凑，临时穿插的事情也特别多。怎么去兼顾大家的时间又能做到阶段性的回顾，让大家及时沉淀，从中有所收获呢？

SM 再一次尝试改版了回顾会。决定将回顾会改良为“简短回顾会”，时间虽短，但内容并不缩水，回顾会会在每个 Sprint 结束后的第二天，站会后立即召开。由于上一个 Sprint 刚刚结束，大家在过程中碰到的问题和吐槽的内容还都印象深刻，这时候蹭热打铁切中问题关键，针对性地做总结会效果更好。由于是站着的原因，时间势必也不会拖得很长。

大家把问题都暴露后，同时也需要把所有问题都落实到解决方案，有的问题甚至有多个解决方案，每个解决方案也会分配具体的 Owner 或团队去跟进。在下个 Sprint 回顾的时候，也需要对上个 Sprint 所有的问题进行检查，这些问题是否都在解决，或是否已解决。

简短回顾会召开了几期后，大家普遍觉得会议效率提升的同时还能起到关键性的总结作用，并能真正解决问题，落到实处。

退改sprint0419&0426				
序	问题描述	解决方案	Owner	状态
1	更换接口，部分功能场景考虑不全，导致开发评估有误	1.产品增加调研	产品经理	已改进
2	新功能上线后的报表监控	1.新功能报表监控需要同步上线 2.prd罗列监控点，具体埋点	丁泉顺	已改进
3	新接口数据不准确	?		close
4	接口挂了，导致delay	与后台沟通，了解测试环境具体原因	陈亮	close
退改sprint0503&0510				
序	问题描述	解决方案	Owner	状态
1	周二封板，周三发布仍然很晚，实际周二没有封板成功	约定周三三点发常规版本回归	开发人员 跟进，测试人员督促	已改进
		周三发布当天发现的问题，提出，大家评估，不随意改动，影响测试常规时间	开发、测试、产品人员 跟进，SM督促	已改进
		遗留的问题，周四早上review	SM发起，全员跟进	已改进
		有新的实验上线后，需要生效后及时check数据，确认分流数据Ok	SM发起，全员跟进	改进中
2	提CP4需要贴图	提CP4需要贴图	测试、产品人员	已改进
3	管伟荣发现的问题怎么落实	发现的Bug，提CP4	管伟荣	已改进
		发现的技术改进，提技改需求	管伟荣	已改进
		产品场景不全的与产品经理沟通提产品需求	管伟荣	已改进
退改sprint0517&0524				
序	问题描述	解决方案	Owner	状态
1	产品经理提的Bug，开发人员改动好未通知测试人员	开发修复好通知测试，CP4加关注（修复之前）	开发人员跟进	已改进
		产品提好Bug，也加关注	产品人员跟进	已改进
		测试人员配合提供产品经理发现的Bug的报文	测试人员跟进	改进中
2	后台发起的改动	涉及的技改提CP4，通知到产品、测试人员， 以及SM，更新任务表	开发人员跟进	已改进
退改sprint0601&0607				
序	问题描述	解决方案	Owner	状态
1	还原度问题	后续产品经理会提改签的改版需求，原型后续直接维护在新的代码上，如老的版本有需求过来产品经理 先过滤，紧急改签需求 涉及老版本样式改动的仍用以前模式调整	PM	已改进
2	线上缺陷排查问题，目前问题多，开发工作会被干扰	1.所有问题都抄送给 TS组	开发人员	改进中
		2.TO过滤问题，排查后与产品经理 确认是否当前sprint修复	TO	改进中
退改sprint0614&0621				
序	问题描述	解决方案	Owner	状态
1	API简化项目临时插入	产品经理加入跟进项目，前期要开始准备	PM	待改进
2	RN项目时间久			
3	火车票、接机票的退改，一旦有调整，火车票需要重新联调	发布通知组加上火车票 部门的人	滕云	待改进

3.2.2 碳酸分享会——让团队都融入其中

“碳酸分享会”是携程机票前台特有的一项传统分享会，已举办了 3 年，在携程的大家庭中一起分享属于彼此最深处的内心世界，建立共同的奋斗目标，营造良好的团队氛围。

作为携程大家庭的一员，我们每个人都有着自己的精彩故事，或是自己的个人经验，或是一些个人想法的总结，或是有意思的事情，无论是工作的还是生活的，这些都可以在这里与大家分享。为了鼓励更多的同事勇于发言，走上台前，形成一个良好的团体分享的氛围，分享会也得到了很多经理和骨干人员的支持。

如何举办

我们希望这个分享会的氛围是轻松的，积极正面的，人人都可参与的。并且在分享会现场都会给大家准备碳酸饮料，所有来宾都可以一边喝着碳酸饮料一边现场聆听，“碳酸分享会”也因此得名。当然，分享议题也不会做任何限制。

嘉宾和主持人可以来自任何不同的团队，可以自己报名参加也可以由领导引荐。报名内容需包含个人介绍、个人照片和团队的信息。

每一期分享会都配备有 2~3 位分享嘉宾及一位主持人，除了分享议题每期不同，我们的主持人也是每期轮换的。主持人负责开场、介绍分享嘉宾、把控整个分享会的节奏等。分享嘉宾则需要在 20 分钟内分享自己的故事或者经验。

分享会结束后，每一位参加碳酸分享会的分享嘉宾及主持人都会获得一份特别定制的礼物及嘉宾合影留念。物质的奖励除了给参加的同事最大的鼓励，同时也是让更多人报名参加的一个吸引点。

我们请了专门的视觉同事帮我们设计 LOGO 和文化墙，每一份礼物都会贴上碳酸分享会的 LOGO 纸贴，每一期分享会嘉宾和主持人的合影都会打印出来贴在我们的文化墙上，让团队成员都融入其中。

期数	日期	嘉宾	议题	团队
第1期	2016.10.28	向阳	工程师文化的团队	机票无线
		刁洪雨	别看见终点，还犹豫不决 ---旅行那点事	H5-产品
		江鹏	带你看日本	服务端
		尹谜眉	主持人	H5-产品
第2期	2016.11.11	李国怀	iOS动画	APP国际机票主流程
		陈伟俊	如何买到便宜的心仪机票	APP X 产品
		吴正菱	主持人	APP国内机票 产品
第3期	2016.11.25	陈小巧	生活离了电影，好比蹲坑没带手机	国际前端产品
		周华明	二八定律到千人千面	前端
		何春刘	主持人	国内产品
第4期	2016.12.9	吴嘉轶	迪斯尼的那些事儿	国内前端产品
		吴嘉敬	视觉导向的理性和感性思维	UED视觉Leader
		王琳	《我是怎样迷晕你的 — 天猫双十一视觉创意学习共享》	UED视觉组
		肖雯	主持人	APP国内机票 产品
第5期	2016.12.22	张晶	聊聊如何写专利那个事儿	机票交互组
		崔庆	上海的一些展馆与展览	自动化测试组
		施晶昱	主持人	国内机票产品组
第6期	2017.1.20	王欣彤	探"女主播"的幕后台前	机票研发部 Hybrid-订单详情
		何立唐	跑马看世界	H5 SCRUM团队
		黄盛华	主持人	后服团队
第7期	2017.2.24	冯卓卓	要事第一	国际产品组
		张轶凡	普通人的奥斯卡-聊聊狼人	机票重构组
		殷潇磊	主持人	机票前台产品订后服务组
第8期	2017.3.10	张富强	你焦虑的原因可能是急于过"标配"的人生	机票无线Android
		吴晓莹	男色当道，如何机智地分辨身边的基佬	APP测试
		黄撞	国际网站比价调价ab测试实践	国际业务部机票数据组
		王卓	主持人	机票后服务增值Scrum Team
第9期	2017.4.7	彭薇	飞行助手中的RN轶事	后服产品Team
		黄玉	地球之71%的世界	机票UED-视觉组
		何春刘	精神分裂的奇妙世界	机票前端产品
		郑开文	主持人	机票H5 主流程

碳酸分享会之特别版

碳酸分享会自 2014 年创办第一期以来，至今已 3 年，文化分享的氛围已深入到团队中。随着碳酸分享会的名气越来越大，也经常会有外部团队人员参加我们的分享会，我们会结合当期分享会的议题邀请到一些神秘嘉宾，或者在其中穿插一些其他活动，举办碳酸分享会之特别版。

1．破万庆功会

机票无线部门在 2014 年年底当日订单量破万，我们在当期的碳酸分享会加入了庆功会环节，除了庆功会提前准备的互动游戏、奖品及定制版的蛋糕，机票部门 CEO 熊星更是这次分享会的特别嘉宾，随着熊星总经理的一番致辞，整个活动现场气氛到达了顶点，让大家的内心都慷慨激昂。这次特别版的碳酸分享会，让大家都感受到了团队共同努力的收获，也建立了我们冲击更高票量的共同目标。

2．捉“虫”荣耀赛颁奖礼

捉“虫”荣耀赛是机票前台部门首次举办的全团队找“Bug”活动，团队间彼此发现问题，探讨问题，共同保障质量。此次活动一共组建了 12 支捉虫小分队，60 人参加。最后大家抓到的“虫子”以及提出的产品建议共达到了近 700 的量级，经过一周的时间裁判进行了严谨的判断与评分，最终第一名队伍以 109 个 Bug，有效分 125 分的高分获得了最终的冠军。

颁奖典礼同样安排在了我们的碳酸分享会上，我们提前准备了“啄木鸟王”奖杯，由机票前台部门的向阳总经理给第一名队伍颁奖。

3.3　让团队更加幸福

凝聚团队，聚焦目标，暴露问题，解决问题，创造持久成长动力才是敏捷的最终目标。

3.3.1　如何培养团队目标感——让团队更优秀

在团队建设中，有人做过一个调查，问团队成员最需要团队领导做什么，70%以上的人回答：希望团队领导指明目标或方向；而问团队领导最需要团队成员做什么，几乎 80%的人回答：希望团队成员朝着目标前进。从这里可以看出，目标在团队建设中的重要性，它是团队所有人都非常关心的事情，有人说“没有行动的远见只能是一种梦想，没有远见的行动只能是一种苦役，远见和行动才是世界的希望”。那么基于

完成自己想要达到的目标所能坚持的毅力和拼劲，就是我们今天要讲的目标感。

所以说：保持 Scrum 团队快、准、稳的运转，与整个团队是否具有高度一致的目标感密不可分。所以到底什么是目标感，如何培养团队的目标感，这是一个一直值得研究和讨论的话题。

事实上，工作中，每天你都经历着与目标感的碰撞。开发人员延期了产品的需求，产品经理嚷嚷说你们评估工时说可以如期上线，怎么到了上线节点就要延期？这似乎是互联网公司常见的产品经理和开发人员碰撞的桥段。事情表现层似乎是开发人员的问题，那底层呢？底层我理解下来大约是产品经理在日常的工作中没有建立起我的项目一定不能存在延期的目标感。产品经理本身没有目标感认知，也没做到目标感价值观传递。久而久之，开发人员觉得你的需求延误了似乎也没什么大不了，于是就发生了上面说的桥段。

评审会上开发人员讨论产品的需求，有些产品经理娓娓道来，把产品设计的缘由说得很透彻，有些产品经理则被问得哑口无言。能说得透的产品经理的目标感更强，他们更懂得他们要让产品在每一次的迭代变更中有怎么样的变化，数据是否可呈现正向发展。说不清的产品经理大多并不清楚在自己功能的叠加中产品本身能获取到什么，自己到底为什么要去做那些需求。所以，目标感是一件时刻围绕着工作每一环节的事情。

不管是对于一个项目的立项、发展还是维稳等环节，PO 是在项目中都起着比较重要的角色，而 PO 也有义务去为 Scrum 团队中的每一位成员建立夯实的目标感。那么，对于如何提升 Scrum 团队的整体目标感，应该如何做呢？

1．项目是大家的，不是 PO 一个人的，既要有责任意识，也要有义务意识

很多 Scrum 团队成员对于 PO 发起的项目更多只是执行者，他们缺乏主人翁意识，更多只是将 PO 的创意编写成代码，在线上跑。作为 PO，首先，你需要做到让大家知道，我们立项的目的是什么，我们要实现的目标是什么，我们每一次迭代能对我们的项目起到什么样的效果；其次，每一次的里程碑，你需要及时同步，自豪感不是 PO 一个人的，是团队的；同样的，产品线上的任何事件，Bug 问题反馈，也需要让团队有共识，责任也不是产品经理一个人的，也是整个团队共同需要承担的责任。

2．PO 想好产品设计的前因后果，做好产品未来半年规划

作为 PO，如果可以经常地分享对于项目未来的规划，尤其是可能对 KPI 影响比较大的项目的底层的设计理由和规划。这样不仅能让 Scrum 成员更清楚地知道未来大家会朝着什么样的方向去做，为什么会这么做；也会更好地调动团队的积极性和热情，将大家拧成一股绳，共同为同一个目标努力。

3．Scrum 团队中的每一个成员共同定义目标承诺，让他们参与目标制订

将整个项目的结果指标拆分成一个个环节的过程指标，每达成一个过程指标，整个 Scrum 团队一起去定义每一个里程碑完成时候的目标承诺，这样会在调用大家积极性的同时，也会非常享受攻破每一个过程指标时候的喜悦。参与越多，越能有主人的感觉。

4．PO 把控产品的走向，团队成员贡献符合其产品走向的需求

在这个人人都是产品经理的时代，谁都可以有好的想法。那么，能让团队成员贡献好的创意，又能让他们体验一把当产品经理的感觉，同时可以凝聚团队的目标感，何乐而不为呢？

3.3.2　持续优化——让团队更出色

在新组建的团队中，在 Scrum 流程还未完全建立起来的时候，SM 为了让团队更顺利地开展工作，会好心地提醒团队要开始开站会了，要更新任务卡片状态了，要按时完成承诺的任务等。但是不在适当的时候慢慢退出，会发现团队越来越依赖 SM，SM 也因此越来越忙，被拖到无尽的监工工作中。

做示范和监督是出于好心，但是这样的好心一旦变成习惯，就会侵蚀团队的自我组织能力，团队得不到该有的成长。在实际过程中如何做到既不干涉团队又对团队有所帮助是一件很难拿捏的事情。对于初级团队，SM 的作用是主导和控制，帮助团队建立敏捷习惯，对于中级团队，SM 的作用是引导和教导；对于成熟团队，SM 的作用是辅导和协助 。所以对于不同级别的团队也要学习采取不同的适合团队的方式，当团队需要你帮助一起决定的时候，一定要先站出来，该坚定的时候一定要坚定，该退出的时候就要毫不犹豫地站到团队后面。

在团队合作过程中遇到问题是很常见的事情，问题是无处不在的，所以欢迎问题的到来比起拒绝问题更容易让团队取得成就感。“问题不在于遇到了问题，相反，问题在于认为遇到问题是一个问题。”——西奥多·鲁宾。

分享一个有趣的故事：

一个老太太有两个儿子，一个是卖伞的，一个是晒盐的。每当天气晴朗的时候，老太太为自己卖伞的儿子很焦虑，因为晴天伞卖不出去；每当下雨的时候，老太太为自己晒盐的儿子很焦虑，因为没办法晒盐了。有一天一位智者问这位老太太你为什么焦虑啊，老太太说出了她焦虑的原因，然后这位智者笑了笑说：“下雨的时候，你卖伞的儿子生意一定很好；天气晴朗的时候，你晒盐的儿子能晒很多盐。”老太太突然明白了，然后很高兴地对智者说：“谢谢！”所以遇到困难、逆境、痛苦时换一个角度也许会有不同的效果。

刚刚接触团队时的我们总是习惯地拿现在的团队和之前的团队模式进行比较，一旦出现与之前的团队习惯相悖的现象，很不幸地就会被拿来讨论，甚至直接把它定性为问题来处理。这也就是在和一个有工作经验的团队新人的沟通过程中总会被反馈一些团队“不合理”现象的根源。

举个例子，在我加入机票部门做 Scrum Master 之前，我是酒店部门的 Scrum 团队中一个成员，我们现在的工作模式为：

1．记录每个人的工作量，定期发布个人饱和度的数据。

2．每个需求测试先写测试用例，然后开发人员自测，测试人员在开发人员提测邮件发出后再介入测试。

3．计划会议上不讨论技术实现方案，讨论提测时间，发布计划。

4．需求不是按照基点估算的，而是按照完成的工作量评估的。

5．一个需求分前端、服务端，测试人员分别评估时间，而且绝大部分还是经理说了算。

6．产品新增变更需求频繁，发布日期一再延期。

7．发布日必须在 18:00 后发布生产。

……

我会认为以上的模式全部有问题，因为这些和在酒店部门我所熟悉的工作模式完全不同。从另外的角度看，记录个人工作量可以更好地衡量团队资源的利用率，提测邮件的发送可以保障软件的提测质量，降低团队过程中的返工成本。先不分析以上模式是不是都有问题，但至少我们不去回避问题。发现问题，暴露问题是一个很好的习惯，但是要避免盲目地解决问题。

分享一个实际操作中失败的回顾会案例，这个案例的主角是我：

> 在进入团队的第二个星期我就主持了我人生中的第一次回顾会，我当时准备了很久，会上也很紧张。我按照之前在酒店部门参与过的回顾会模式，SM 总结了团队的现状，做了一次 Sprint 的复盘活动，然后采用提问的方式让团队成员分析各节点的问题并给出相应的解决方案。一切好像都按照我正确的方法进行着，会议上讨论很激烈，会后也有了产出，全程看着都很合理，很顺利。然而当时回顾会上的气氛是压抑的，呈现出一种批斗会的感觉。那样的会议氛围是超乎我预料的。明明都是按照标准步骤走的，但是就有一种走错的感觉。

后来的事实也证明我确实走错了，会后我看到了大家对于回顾会的抵触，以及对于我的不信任感。

以下截取了当时回顾会的部分概要内容：

1．问题：新增的紧急需求因产品经理和研发人员理解不一致，导致发布延时。解决方案：后续加强多方沟通，明确需求功能点，准确评估时间，保证发布顺利，大家一起进步。

2．问题：需求提测延期导致测试压力大。解决方案：明确需求优先级，测试工作量大的需求前移，避免测试后期压力太大，在计划会上大家一起做好排序。

以上两个问题描述相信大家在平时的工作中或多或少都有遇到过，以上的解决方案也非常眼熟，但是这些解决方案我们后续无法考量，无法验证是否得到了改善，只有等出现相同的问题的时候才能验证问题是否得到了改进。事实是那些加强沟通，明确需求的改进方案对于解决根本问题毫无帮助。项目过程中很多问题都能被追溯到同一个根源，那就是在某些重要事情上沟通不到位，没有达成一致。所以冷静下来，认真反思，透过它的表象找到一个或多个潜在的起因。

在团队再次因为新增任务导致发布延期的时候，我找了团队成员一起分析了根本

原因：因为项目组织的特性，团队对于发布的日期没有一个很严格的要求，所以当遇到 Sprint 阻碍的时候团队优先选择的是推迟发布日，而推迟发布日的结果就是后续的 Sprint 计划一直在变，大家不得不频繁地中断手上的工作，举行临时性的计划会。对于团队来说，有规律的节奏感可以帮助他们了解自己的速度。大家也非常愿意能更好地掌控好自己的节奏，于是一致商量后决定发布日期不轻易改变，针对新增需求、紧急发布等采用置换需求的方案保证发布的顺利。新公约制订执行后，团队发布的准时率提升到了 90%。

最有用的教训是从错误中学来的，不要着急地去解决问题，改变是需要耐心等待的，不能急于一时。要允许团队犯错，只有犯错才会更好的进步。当团队频繁发生相同问题的时候，冷静地分析问题的根源。不要为团队做太多的决定，把问题交给团队，相信团队才是最好的问题解决者。

及时地暴露问题也是团队持续进步的一个重要方法。“这个在特殊数据测试环境下没有的嘛！”“这套环境在每周某个时间都会挂掉！”“某某开发的程序 Bug 好多啊！”……这样的抱怨一直伴随着我们的工作，以至于新人来问这些问题的时候我们习惯性地告诉他们，“淡定啊，一直是这样的。”这些影响我们效率的问题常常被隐藏起来，我们慢慢地养成了惯性思维——这些就是改变不了的。

进行 Scrum 后，这些问题就是 Sprint 周期内的障碍项，慢慢地就被透明出来，引起了重视。并得到了很好的解决。

1．开发提测质量差。这是团队对于质量的一种态度和规范制度建立不良导致的。问题的根源：测试人员未提供相应的自测用例和开发人员的意识中质量是测试人员的事。针对此，团队要灌输质量是大家的理念，最终可交付的成果是大家一直需要去保障的思想，并设定了落地方案，测试人员提前给出冒烟测试用例，开发人员全部通过冒烟测试后才进行提测。经过一段时间的磨合后，来来回回的返工成本降低，团队的默契度提升，团队成员之间的信任度也提升了一个高度。

2．构建测试数据的成本太高。这个问题是当时影响团队测试效率的一大拦路虎，属于测试过程中的一个大痛点，慢慢地也变成了 Sprint 失败的一个主因。一开始的解决办法是收集构造困难的数据需求，定期维护，但这样会增加长期维护数据的工作，再后来定期收集数据痛点，统一维护，数据共享，最后集中攻破。我们搭建了数据共享平台，在半年的时间内构建测试数据变成了一件轻松简单的事情，测试的效率明显

提升，测试人员的幸福感提升，Sprint 的完成度也相应达到了稳定的趋势。

3.业务逻辑只知其表，不知其里。线上问题排查曾经在某段时间是阻碍团队 Sprint 正常进行的最大祸首，该问题反映了线上质量的不稳定以及定位线上 Bug 的能力水平。关于 Bug 的定位能力又反映了两个问题：排查的方法是否正确；排查人对于业务的需求是否理解深刻。团队做出了以下的改进：开发排查日志小工具，方便查找日志；设定业务专家，梳理需求数据流向以及逻辑脑图。后来定位问题的耗时率下降了 50% 左右。

4．环境不稳定。联调或者测试到一半，突然所有的业务数据不见了，因为别的团队改动的代码导致 Block 情况反反复复，而明天项目就要上线了，这样的状况着实恼人。针对这个问题，经讨论每个 Scrum 团队设置了专属的环境，因为环境不稳定导致的 Block 率也大大下降了。

我们变得更好了吗？实时地衡量自己的团队，还在不断突破自我吗？不害怕犯错，大胆地犯错，只要我们一直在持续改进的路上，就是一支高产出的团队。养成把困难看作是一次次获得锻炼的好机会，只要我们和困难做了斗争，我们的能力就会提升，团队满足感和幸福感也会相应提升。

第 4 章

极致敏捷的小团队

在创新、创业高速发展的时代，一个小业务部门如何发展成为一个大的业务群，带来 *N* 倍的业务增长，在这个背后，极致的敏捷团队起着无可估量的作用。在 CEO、中层、一线经理眼中，这是一套相较于一般敏捷团队更敏捷、更极致的玩法。

4.1 业务连年 *N* 倍增长背后，有哪些创新的管理方法

携程火车票 SBU 部门这四年来创新创业高速发展，形成地上交通业务群。这里面有哪些创新的管理方法？如何高效驱动业务增长？如何在发展壮大的同时保持高效，避免大公司病？如何激活团队的创业激情？如何培养和管理创业型人才？如何塑造创业文化和避免稀释？……如果你的组织遇到类似的问题，我们的系列实践或许能给你参考。

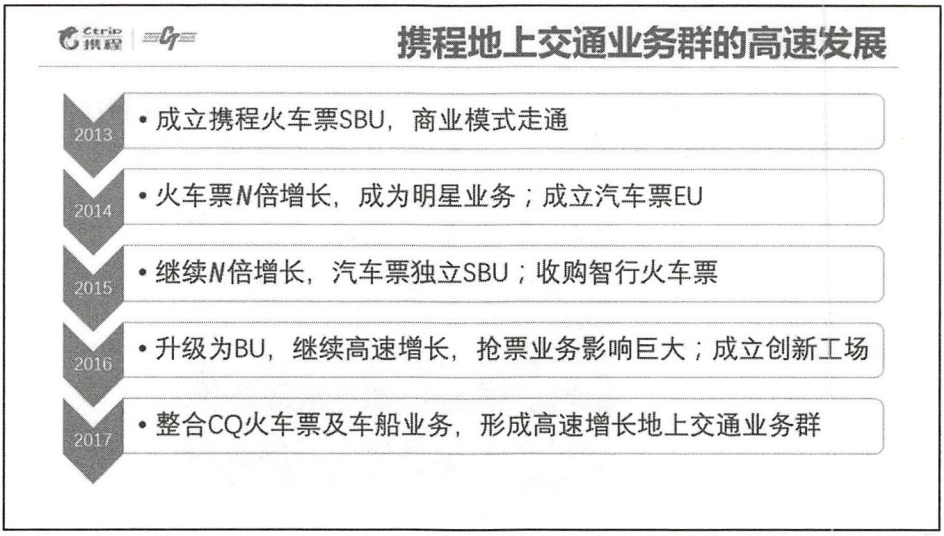
1．创新的组织结构：OK 制

2．高效的运营管理：OKR

3．创业型激励机制：投名状

4．人才培养和管理：黄埔训练营

5．文化塑造和改造：管理层理念革新

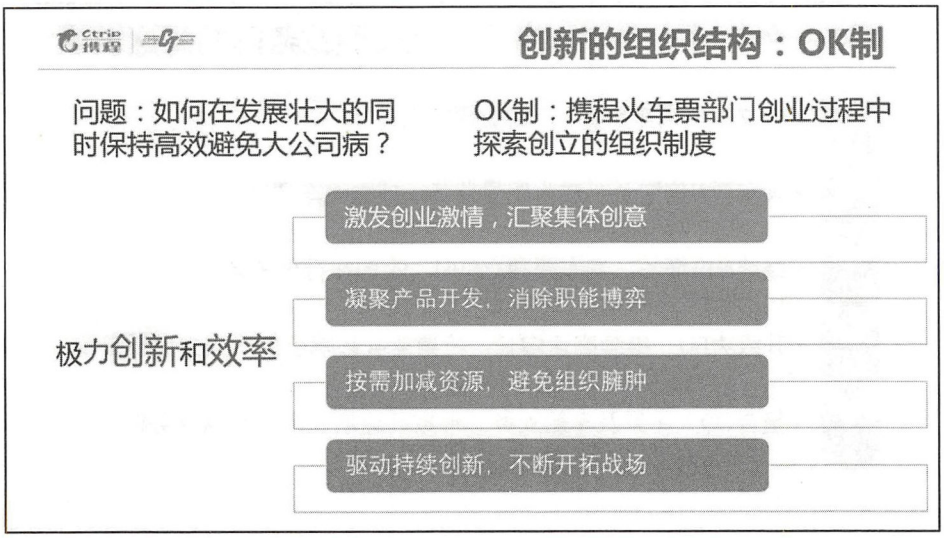


4.1.1　创新的组织结构：OK 制

创业团队通常都是小团队，很敏捷，随着业务发展和团队壮大，大公司病就来了。常见的病症有以下几种。

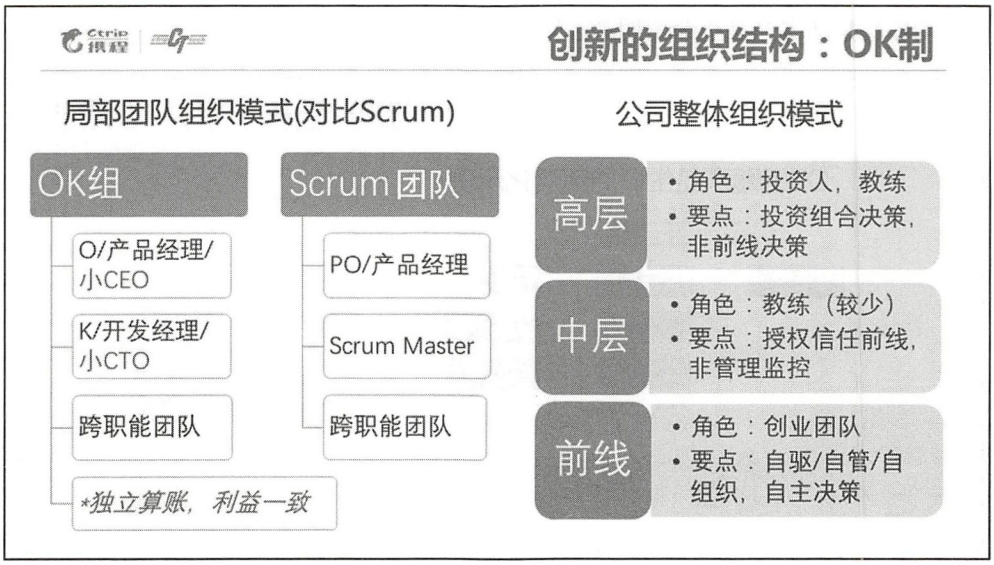
- 基层是螺丝钉，中层是监工，高层做决策。基层的角色只是执行，中层上传下达监控基层有没有认真干活不偷懒，高层是组织驱动力的来源、创新的来源和决策的中枢。这种结构基层很无聊、中层很烦、高层是瓶颈很累，组织走得很慢。
- 产品部和开发部互相博弈，沟通效率低，合作成本高。两边通常有各自的目标，遇到点事就吵。例如技术改造这点事，产品经理吐槽开发人员没事找事耽误事，开发人员鄙视产品经理什么都不懂。甚至升级到两边经理吵，两边经理吵不清楚，升级到两边总监吵……
- 随着业务的发展，人越招越多，当业务进入成熟期，没什么大需求了，这么多人没事干怎么办？产品不断提 CR 修修补补，开发人员不断重构代码，ROI 是不考虑的，人不要闲着才是重要的。这种情况看起来大家做的事情都有价值，其实价值很小，甚至无法覆盖人力成本。

OK 制是携程火车票部门创业过程中探索创立的组织制度，能在组织发展壮大的同时保持高效，避免以上各种大公司病。



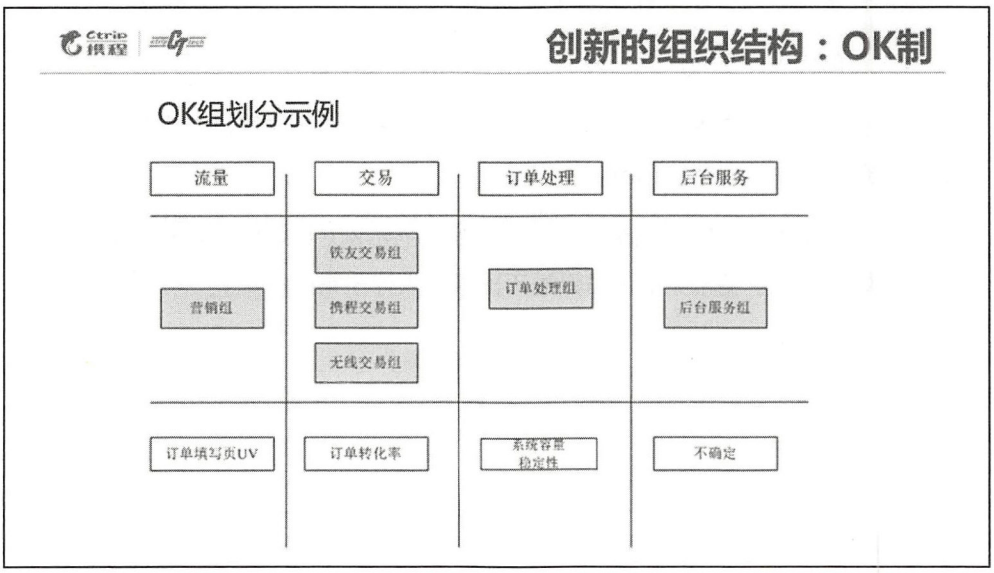
OK 制在整个组织表现为一个个 OK 组，OK 组在模式上和 Scrum 团队有同有异。OK 组一样有三个角色：O 代表产品经理，K 代表开发经理，加上能做独立交付的跨职能团队。OK 组没有 Scrum Master 这个角色，因为 OK 组是纯粹自我驱动自组织自管理的。

另外 OK 组的角色是创业团队，O 是小 CEO，K 是小 CTO，O 和 K 共同决策，不同意见以 O 为主。不需要上级总监或老板来下达作战指令或审批决策，中层很少，主要角色是教练，而非管理监控者。高层除了教练主要是投资人角色，做投资组合决策，如往各个 OK 组投多少资源。高层和中层会对 OK 组的决策进行 challenge/教练式提问，为 OK 组决策提供补充信息和思路，促进 OK 组提升决策质量，但不会代替 OK 组做决策。

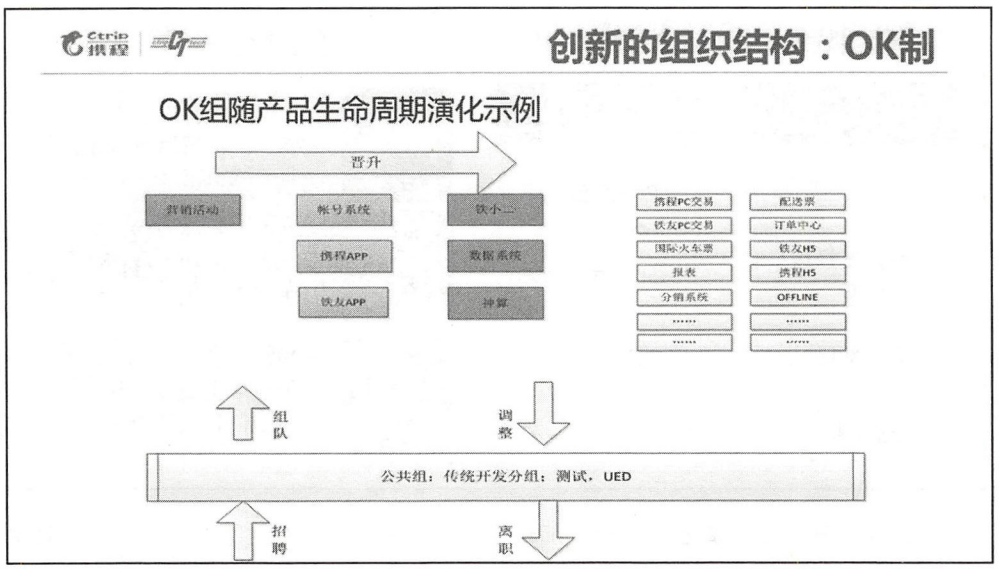


OK 组一般能独立算账，升职、加薪、发奖金首先看 OK 组的业绩，其次才是组内不同人员对业绩的贡献。所以组内不同职能都具有高度一致的利益，更易精诚合作，同时自主决策比等高层决策能更敏捷适应市场和行业环境变化。

下图是 OK 组划分的一个示例。创业早期只有一个 OK 组，随着业务发展团队壮大，为了保持高效，业务拆分模块、目标分解指标、大团队裂变成多个 OK 组，OK 组的人员规模保持与 Scrum 团队规模相当，一般几个人，最多十几个人。



OK 组是随着产品生命周期动态演化的，如下图所示。



红色代表产品启动期（从 0 到 1）。从资源池抽取几个人成立 OK 组，搭建和验证产品。

黄色代表产品建设期（从 1 到 N）。根据产品路线图推进，从资源池卷进越来越多资源到 OK 组。

绿色代表产品运营期（从 N 到 N+1）。释放 OK 资源，合并公共资源。

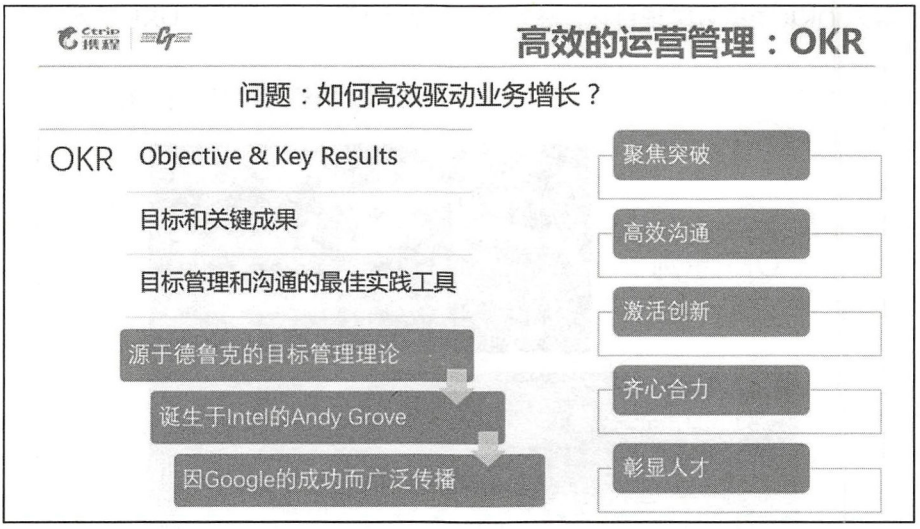
右边白色代表产品收尾期。及时合并/关闭低产出项目，释放公用资源。

如果公共资源池变得富余，则指示我们需要开拓新业务或突破老业务瓶颈，形成新的增长点，驱动大家持续创新。

下图是两个 OK 组随产品路线演化人员配置的示例。OK 组的人员规模敏捷适应业务的发展变迁，避免常见的人员只进不出最后人浮于事的情况。

4.1.2　高效的运营管理：OKR

OKR 是互联网行业新的目标管理方法，也是高效的沟通工具。其思路源于德鲁克的目标管理，德鲁克的粉丝 Intel 的总裁 Andy Grove 发明并推行了 OKR，Intel 的 John Doerr 将 OKR 带给了 Google，随着 Google 的成功实施，OKR 方法被其他知名 IT 企业借鉴。



常见目标管理的乱局有两类：

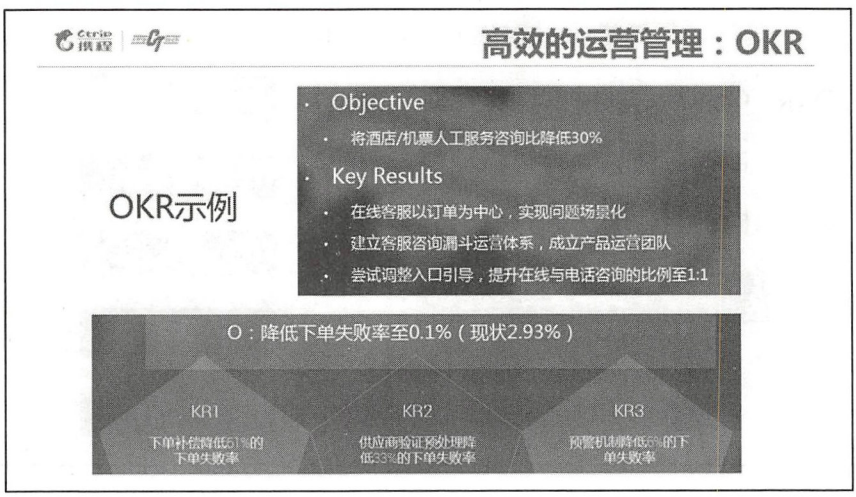
1．缺乏目标的混乱

- 尤其是研发部门，有的研发部门没有自己的目标，产品经理说做啥就做啥。或者团队成员没有目标，等待领导指令，结果领导很辛苦，员工很悠闲。
- 有些产品也只有粗略的方向，没有有效的目标。杂七杂八的需求做了一大堆，耗费很多资源但看不到突出成效。
- 不知道公司战略和部门目标造成方向性浪费。例如，产品经理决定后续停止 Online 站点的运营，研发人员不知道，还在安排 Online 技术改造或自动化测试开发。

2．KPI 管理的弊端

- 一切为了 KPI 和绩效奖金，造成低承诺，高实现，甚至不择手段。例如，为了代码行的 KPI 制造冗余垃圾代码。
- 不同职能 KPI 对立导致争执冲突内耗。例如，开发人员以 Bug 数作为代码质量 KPI，测试人员以找出 Bug 数作为效率 KPI，开发人员和测试人员为了一个问题是不是 Bug 吵个不停。
- 只见树木不见森林，为了 KPI 忘了初心。有个笑话很应景：一个人沿路不断挖坑，后面跟着一个人不断填坑，这个奇怪的事情背后是中间种树的那个人请假了，这种情况完成挖坑和填坑的 KPI 而不顾种树的初心目标只是浪费资源。

导入 OKR 能有效管理目标，高效驱动业务增长。下面是两个 OKR 的示例。



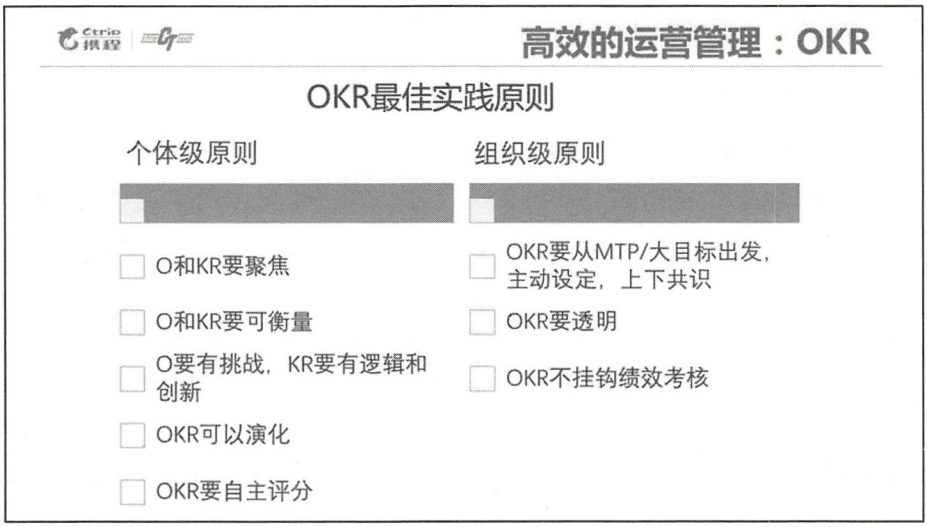
参考示例，我们来看看 OKR 的最佳实践原则。

1．个体级原则

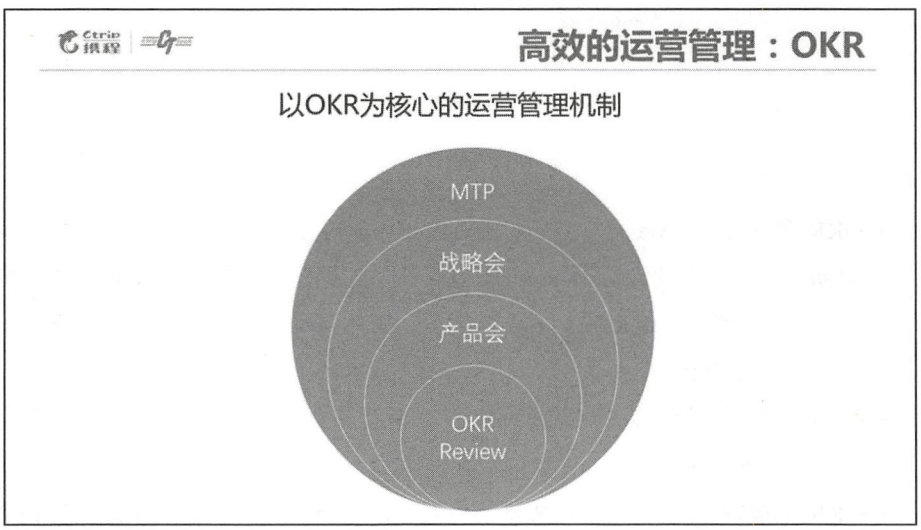
- O 和 KR 要聚焦，个人一般 Own 一两个 O，部门负责人可能 Own 三四个 O，只有聚焦才易突破。每个 O 的 KR 一般也不超过 4 个，抓住实现 O 的关键点。
- O 和 KR 要可衡量。这点与 SMART 原则是一致的。“提升支付成功率”这样只算方向不算目标，是不可接受的。
- O 要有挑战，KR 要有逻辑和创新。OKR 鼓励和驱动大家突破自己的能力，突破业务的瓶颈，有挑战性的 O 才是合格的 O。通过 KR 的逻辑和创新，支撑挑战性 O 的实现。
- OKR 可以演化。这个世界变化越来越快，年初制订的 KPI 年末很可能因为环境变化而完全不适用，甚至从季初到季末都存在刻舟求剑的问题。所以 OKR 可以随着环境的变化或信息的变化而演化，可能提高目标也可能降低，但原则都是保持目标的适度挑战性。
- OKR 要自主评分。通过评分来促进进展盘点和沟通，评分标准要严谨，结果高一点低一点不重要。

2．组织级原则

- OKR 要从 MTP（Massive Transformative Purpose，宏大变革目标）/大目标出发，主动设定，上下共识。OKR 不像 KPI 那样由上级分解指派，OKR 主要靠自己设定，以上级的 OKR 作为方向性指引，上下沟通达成共识。这样在聚焦的同时激活集体创新。
- OKR 要透明。每个人的 OKR 在组织内是公开的，这样容易达成互相了解和齐心协力，避免冲突。
- OKR 不挂钩绩效考核。OKR 基于自我驱动和自我实现，鼓励挑战，挂钩绩效考核会扼杀这些内在动力，所以不能以 OKR 的评分作为绩效考核的评分。绩效评估应另做设计。但是 OKR 可以作为沟通工具高效阐述业绩成果和逻辑创新（经常看到绩效自评列有冗长难读的流水账或“认真负责团结合作”的水话）。



以 OKR 为核心的运营管理机制参考下图，每个圆圈都是迭代循环。

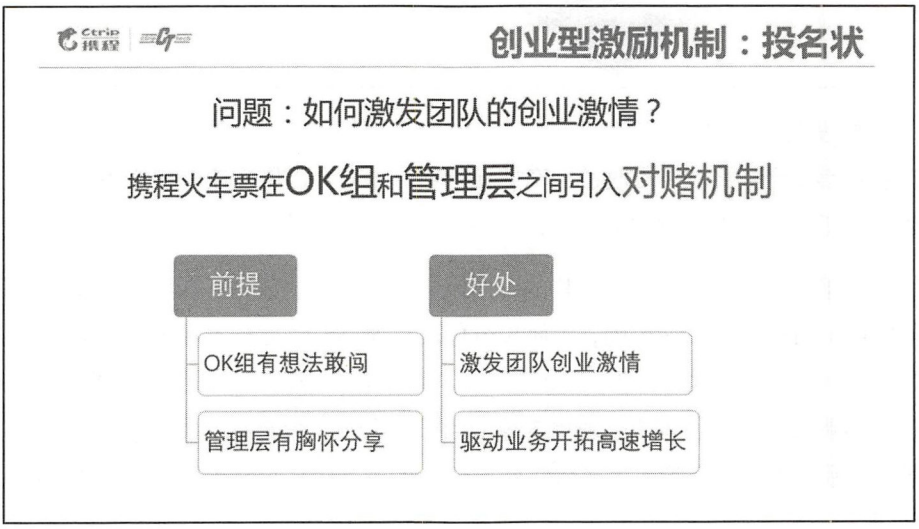


- 外层 MTP 迭代。一般是老板塑造的愿景，例如携程火车票，为全国人民买火车票。
- 往下是年度战略会。一般是总监级阐述年度战略 OKR。
- 产品会是每两个月一次（根据业务节奏也可能调整到一个月或三个月一次），主题为 OK 经理的短期 OKR 设定和阶段性运营盘点。
- 内层迭代是两周一次的 OKR Review 会议（根据业务节奏也可能一周一次），主题为 Review OKR 的中途进展。

中高层教练工作主要在 OKR Review 会议和产品会，高层投资组合决策主要在产品会。

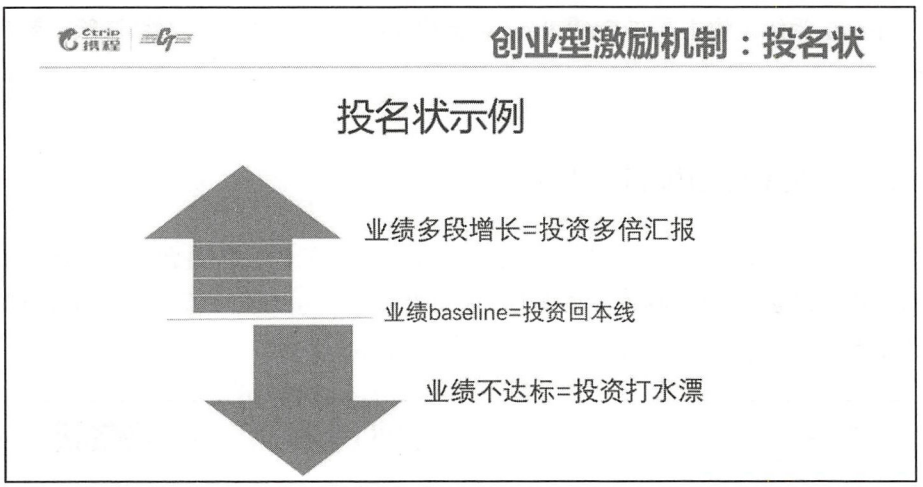
4.1.3　创业型激励机制：投名状

如何让员工像老板一样全心全力创新奋斗？这个问题经常遇到的反问是员工如何像老板一样拿到高额回报？OK 制把 OK 组作为小创业团队来看待，在 OK 组引入创业型激励也是相得益彰的。



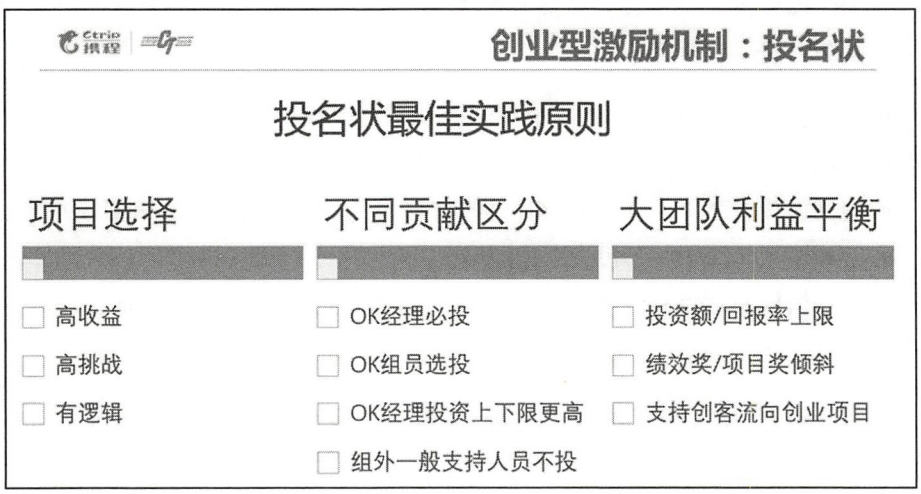
下图为投名状示例，OK 组成员与管理层谈定一个周期内的业绩基线 N，同时 OK 组成员投资 M 元，如果周期末：

- 业绩没到 N，OK 组员亏掉 M 元。
- 业绩到 N，OK 组员拿回自己投资的 M 元（不赚不亏）。
- 业绩做到 N 的 1.1 倍，OK 组员拿回 2M 元（投资 3 万元就是拿回 6 万元）。
- 业绩每超基线 10%，投资回报加一倍。
- 业绩做到 N 的 2 倍，拿回 10M 元（投资 3 万元就是拿回 30 万元）。



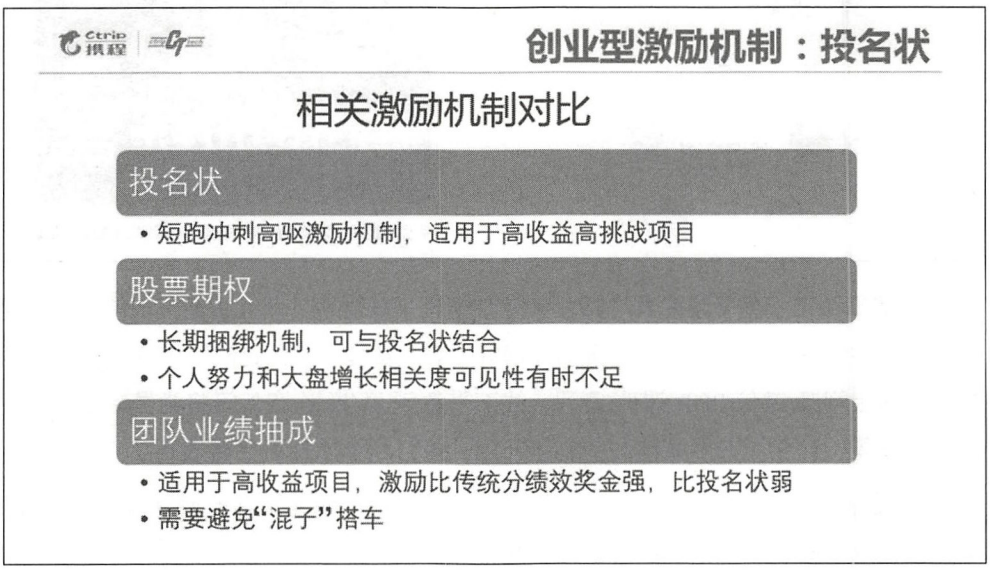
实施投名状的项目大家都非常投入，结果也都获得了很大的成功和回报。但不是所有项目都适合投名状，高收益、高挑战的项目才适合。另外这个挑战还是基于一定的逻辑分析的，没有依据的浮夸风目标是浪费沟通和管理成本。

OK 组不同人的贡献是不一样的，所以会设计一定的区分。OK 经理作为小团队的 CEO 和 CTO 是必须投资的，不愿投资代表对项目没信心和不愿承担风险，那就需要换领军人。作为 OK 组员则可投可不投，同时 OK 经理可以投资更多的额度，OK 经理承担更多的风险和可能获取更丰厚的回报。组外一般支持人员不投，视作乙方合作管理，我们的实践经验是太多合作方投资进来容易扰乱职能本分并引起其他项目合作纠纷。



因为有些团队适合投名状，有些不适合，容易造成大团队的利益不平衡，为了避免贫富差距过大而出现问题，尤其是同等技术能力的工程师在业务团队和支持团队的收入差距太大，为此我们设计了一些平衡规则。包括投资额设上限，回报率设上限；实施投名状的团队不占用 A 等绩效奖金名额，传统项目奖金也留给非投名状团队；同时支持能力强、有创业精神、愿意承担风险、争取更多回报的人流向创业型项目。

当前互联网公司大多实行股票期权激励，几年生效的股票方便将优秀员工与公司长期绑定，有时也会发生个人努力和大盘增长相关度等难以看见的问题。像携程这样几万人的公司，除高层外的个人奋斗也难以看到对大盘业绩和股票的影响，所以总会有人停留在舒适区，仅仅本分工作等待股票生效。投名状机制是股票期权机制的黄金搭档，有效补充短期高强度激励，让大家像创业合伙人一样投入心力、脑力和体力。



4.1.4　人才培养和管理：黄埔训练营

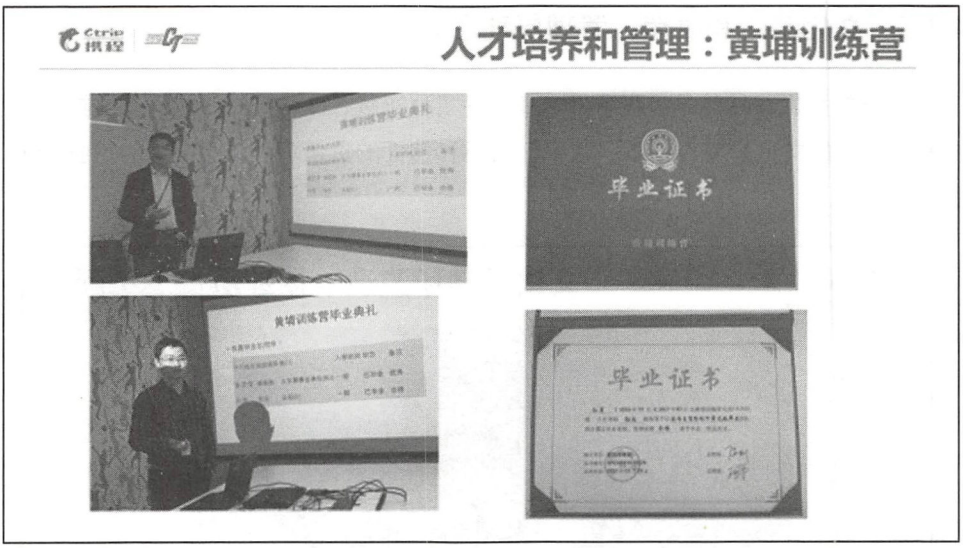
OK 制和 OKR 作为携程地上交通业务群运营管理的核心和特色，与投名状一起驱动着各业务单元高速增长。随着业务增长和地盘扩大，团队也不断壮大及换血，而保持高速增长需要将优秀的管理理念和实践经验批量复制给团队新人，黄埔训练营应运而生。

黄埔训练营的招生主要面向产品经理和开发经理（OK 组的 O 和 K）。当前课程以 OKR 训练为主，同时涵盖了创业和创新、MTP 和 OK 制的讲授，期望学员通过课程掌握目标驱动和业务推进的知识和技能，拥有共同的沟通语言和管理方法，以便高效地协同推动各业务线，开拓创新，实现业务高速增长。

黄埔训练营根据学习发展心理学和培训管理规律独创“多元进阶”训练法。与现在流行的在线交互英语教学、塑身训练营等一样，黄埔“多元进阶”训练法，通过有效利用线上视频/微信/PPT 课件/测评、线下受教/观摩/模拟/实战等将学员轻松带入真实训练场景，让学员在入营前后的创新知识、能力和意愿都有显著提升。

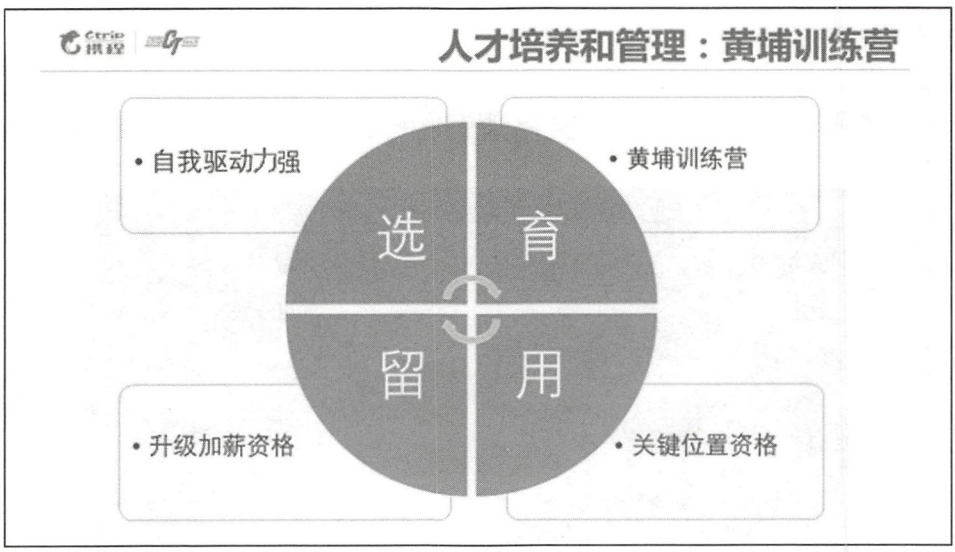
黄埔学员不像传统培训那样上完课就完事，需要完成实战项目后进行毕业答辩，答辩通过才能拿到黄埔总教练签名颁发的毕业证书。下图是两位总教练——携程集团高级副总裁陈刚先生和副总裁王玉琛先生在黄埔训练营毕业典礼上讲话。

黄埔毕业证书不仅仅是个荣誉，还是关键岗位的资格证（OK 组的领军人）和升级加薪的资格证（项目做得好但是 OKR 逻辑阐述差，被视为通过运气成功而难以复制，可以多发奖金但不能晋升）。



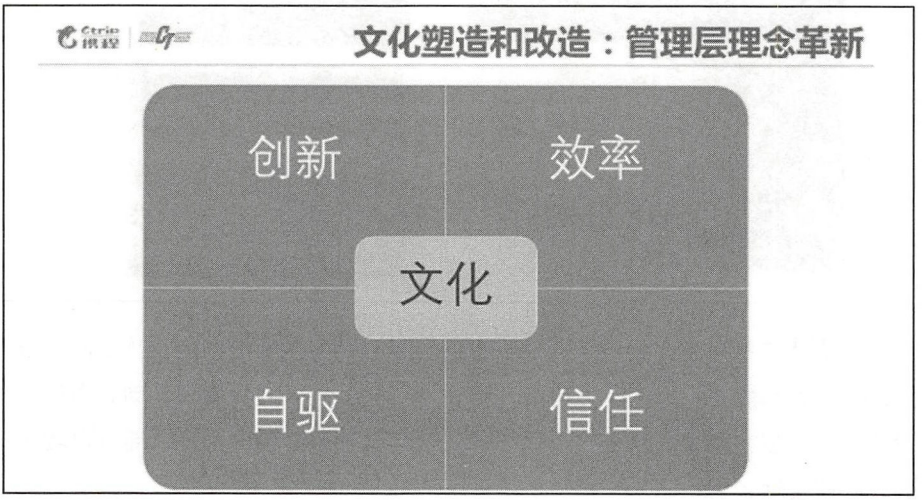
创业型人才不单单是培养出来的，大前提是招募自我驱动力强的人，而不是仅盯着知识、经验和能力。即使能力有限，作为 OK 组普通一兵，自我驱动力强的人不需要设置经理岗位去监督跟进每天具体工作，这会省下很多管理成本，而且自我驱动力强的人的成长性也大。

所以创业型人才的培养和管理，是以招募自驱者为前提，文化和方法论培训为重点，形成选、育、用、留整个环。

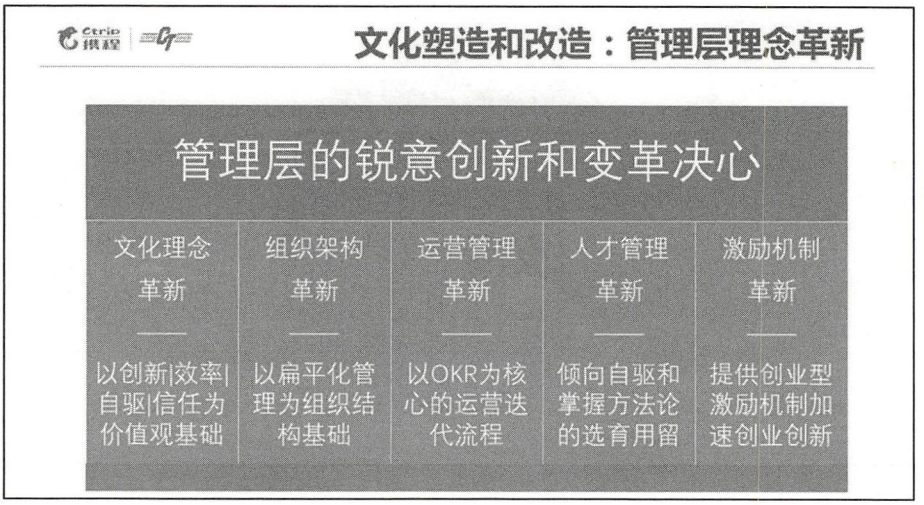


4.1.5 文化塑造和改造：管理层理念革新

敏捷转型最容易的是流程导入，其次是工程实践，最难的是文化改造。携程地上交通业务群的文化是基于创新、效率、自驱和信任的价值观。这也是 OK 制和 OKR 的存活土壤。



组织文化塑造和改造取决于管理层，尤其是老板。火车票团队是地上交通业务群中最早的创业团队，文化塑造相对容易。后期不断发展不断招人，不断孵化新业务裂变新团队，不断整合新业务团队，文化的传承和改造需要新团队管理层的锐意创新和变革决心。



创新、效率和自驱的价值很容易得到认可，而管理层能否信任前线团队让其自主决策敏捷应变却不容易。管理层需要放下理性的自负，相信前线团队更敏锐，集思广益胜过诸葛亮，不再事事决策审批。

从传统复杂的层级职能组织结构转型为扁平化的 OK 制组织结构，需要中层的意识角色转变。从管理者变成教练，更需要高层的魄力：这个过程会有优秀的人离开，也会有优秀的人留下来，还有更多优秀的人被吸引进来。

导入 OKR 为核心的运营迭代流程是改变最容易的环节，人才管理革新则是周期最长、最需要耐心的环节，尤其是资源紧缺时坚持招募标准。而激励机制的革新则需要管理层乐于分享利益的胸怀和格局。

文化是虚的，不容易看见，但是仍然有办法传播。下图是我们汇编的内刊，不断总结管理思路和实践，形成文字，人手一本，方便传承组织文化。

4.2　看 CEO 如何实现一个小目标

CEO 的大目标即组织的最大目标，我们的大目标可能是别人的小目标，但那不重要。我们边做边总结出了火车/汽车票运营的核心，OK 制和 OKR。

4.2.1 OK 制基本运营思路

OK 最早形容一个小组，O 是产品经理，K 是开发经理，他俩合作搞定一切。OK 取自湖人队的 OK 组合，它的基本玩法是发现优秀的 O，给他一块地盘，配以 K 及技术产品资源，形成围绕 O 决策的小组，小组自己决定做什么，怎么做，什么时候做，OK 说了算，出票量或者收入，只要占一头就行。

地盘上的 KPI 制订监控、优化等运营也是 O 来做，类似一个小公司的 CEO，是个相对独立的小组。OK 组是动态的，当这块地盘不能再继续新的产出，没有新的故事，那 OK 组就解散，去抢新的地盘。这项制度随着火车票部门的创业，已经有 3 个年头了，从最早的 OK1.0，它解决了资源不足，打破了传统公司麻雀虽小，五部俱全的弊端。OK2.0 则演进为多个 OK 组管理下的生命周期模式，描述了每个 OK 组如何成立，如何评估，如何激励，如何解散的全过程。OK 制是创业过程中自发发展出来的体系，没有现成的理论和实践指导，只能边做边总结，它的发展是基于以下条件的。

铁打的携程平台，流水的 SBU

携程是中国最大的旅行交易平台，未来努力成为世界最大的旅行平台，平台一旦形成，规模效应带来的稳定性和门槛是很高的。但移动互联网缩短了产品型公司的生命周期，以前一个产品公司的生命周期是 5~8 年才慢慢消亡，现在只需要 3~5 年，而且一旦衰退，很快就消亡。携程实行 SBU 制度，也是基于产品的划分，SBU 的诞生、成长、高峰和衰退也是由产品的生命周期决定的。成功的平台可以大量、持续地诞生 *N* 个成功的产品，从成长到衰退，繁荣不息。所以说铁打的平台，流水的 SBU，再细化的产品局部，比如火车票的 PC 版本和 APP 版本，也是有生命周期的，新的产品形态不断出现，老的从维持到衰退。认识到生命周期的特性，对理解 OK 制的动态性至关重要，产品型公司做得再大，也是一个大型的创业公司，因为产品的生命周期很短。

顶级的产品经理是胜负手

互联网产品是虚拟产品及服务，产品经理决定了它的成败，通常就是 1~2 个人，他能打败大公司。所有同段位的竞争对手，手上的硬件都是一样的：资本，流量，技术人员，供应商资源。唯一差异的就是产品经理，他是胜负手，稀缺资源，找到那个最好的产品经理，并给他足够的资源，这是公司应该做的事情。

扁平化管理的基础是价值观

OK 制将传统的技术部和产品部打烂，传统负责上传下达的中层干部在这种体系里发挥不了作用，有时甚至是负作用，这种扁平化带来高效决策和执行的同时，也带来无序管理的难题。管理大多时候是为了解决两个问题：我说的你做了没有，有没有偷懒？我说的你做得好不好，不要骗我？这是一种基于不信任原则衍生的管理方法。

如果我们聘用一位总监来管理我们全部的 OK 组，他一定很慌，他都不知道手下人在做什么，因为他们不会汇报。所以唯一的办法是我们的管理者必须是 OK 制的坚定推行者，核心人员都有一致的价值观，将需要管理的内容降低到 0，认可并接受这种适度混乱的、自我驱动的状态，通过参与到 OK 组的形式去了解每个组，公正地判新 OK 组成员的绩效和能力，以及更愉快地信任他，与他合作。价值观不是挂在墙上的口号，我们说的价值观是对一系列办公室工作内容的看法，比如：嗓门大一点有没有关系？不开会行不行？只看 OKR 不看主管评价行不行？坐席怎么安排会有利于沟通？决策争论不下时听谁的？会说有多重要？拍马屁好不好？核心人员在这些问题上的标准是趋同的，管理将变得异常简单。

基于这样的考虑，我们设定了动态组织结构 OK 制，像拍电影一样，对每个地盘，我们可以命名为一个电影系列，比如出票组将长期是我们的一个核心地盘，可以命名为“星战系列”，每一段时间都要拍一集。O 是导演，K 是主演，如果 O 导不下去，没有剧本（剧本是 OKR），那 OK 组解散，大家去拍新的系列片。有的 OK 组年复一年有新片拍，有的只剩下回忆了。

2015 年我们找到了一个比较好的工具——OKR（Objective and Key Results）。OKR 用来作为剧本格式描述了你下一部电影要拍什么，主要的目标是什么，怎么实现这个目标，难点在哪里。一开始这个工具用得还不太好，但进步还是挺大的，在口头上体现得更好些，要落到面上，才华有些不够，今后会加大这个训练量。

另外一个工具是月度产品会，每个月，所有 OK 组都要在会上检查各自的 OKR 完成情况和 KPI 运营情况。这大概是我们除周会外唯一固定的会议。产品会特别重要，在 OK 制下，会说自己做得多好实在是太重要了，这叫营销，所有好的电影也要去做路演和推广，不是么？

所以我们团队里最如鱼得水的人就是适应 OK 制，并在里面做出成绩的这帮人，我自己也是。我放在其他部门，可能做不出这个成绩，只有在火车票部门，我才能又

出成绩又开心，我哪也不去，我常常这样想。所以我们今年的工作虽然比往年的挑战都要大，但比往年更加快乐，成绩也好。

2017 年的业绩我不是很担心，事实上我们几乎从来没有就业绩来讨论应该怎么提升它，业绩一直只是一个预测数字，我们讨论的永远是产品、营销、服务！2017 年仍然如此，OK 制的实际运营已经跑到我总结的体系前面去了，我们明年要做的是坚持这种思路，以产品第一的态度来配置资源，以 OKR 为标尺来竞争地盘，招聘与晋升。当我们把这种体系玩得更流畅时，我们的人才团队也会平台化，我们做的产品会变成有生命周期的作品，无论是何种产品挑战，我们都能做到一流，我们的 OK 核心会变成最优秀的导演和明星，而不是成为技术官僚，我相信在平台下的创业，OK 制是很有前途的模式，因为 OK 制发挥了每个人最大的特长，而且是非常快乐地发挥。

4.2.2 首先要有个大目标

我们讲了三年的 OKR 方法论，基本思路是 MTP→OKR→Abtest。MTP（Massive Transformative Purpose）一词来源于《指数型组织》，我只是借用了这个词。总的来说，Abtest 是公司在推，效果显著，OKR 是部门在推，效果一般，MTP 是属于被忽略的，“大目标”经常被批评为假空洞，很主观，但从逻辑上讲它是最重要的。

在我们的组织中，OKR 是自下而上的管理方式，每个 OK 组自己定义目标和实现的逻辑。但从管理层的角度来看，OKR 的存在和 KPI 的分解是一样的，整个部门的 OKR 是为管理层的 OKR 服务，部门存在一棵自上而下的 OKR 分解树，和 KPI 自上而下地逐级分解逻辑是一样的，只是 OKR 树的逻辑是神经传导系统，只传达逻辑和方向，KPI 树则是传递执行命令，有本质的不同。我们经常看到下面组织的 OKR 显得过于零散，并非 OK 组的错，而是很多人看不到整棵树的逻辑，这棵树的逻辑核心在 MTP，也就是顶上的 OKR。因此，向下管理和向上管理的核心在于把 MTP 讲透彻，这是 OKR 树向上追溯“为什么”的方向，我们在年初要把战略会开透，也是这个道理。

我们的大目标可能是别人的小目标，但那不重要。CEO（泛指各种组织的头都叫 CEO）的大目标即这个组织的最大目标。如何制订 MTP 呢？我每年制订年度计划时，问三个问题：

要发生怎样的变化，这个产品才叫发生了质的变化?

要发生怎样的变化，这个团队才叫发生了质的变化?

要发生怎样的变化，这个公司才叫发生了质的变化?

三个问题想清楚了，MTP 就出来了。举个例子，2016 年的火车票在产品形态上发生了质的变化，从一个代购为主的火车票应用转为抢票为主的变化，2017 年如果做成以解决方案为主的，或者多模式搜索为主的，那就会发生质的变化。所以回到前面说的，CEO 的 MTP 是顶层大目标，好多 CEO 给个财务指标，然后一级一级分解下去，这在执行过程中，是实行不了自下而上的 MTP/OKR 体系的。

在我的管理体系中，能推动 MTP 的人是有强烈成就驱动的核心人才，也是 BU 制的核心，作为集团平台，找到这样能提出和驱动 MTP 的人，是 BU 制的基础。在地上交通业务群，我们看到少数这样的精英，具备 CEO 的潜质，我要做的就是给他们搭好舞台，给块地，充分授权，让这些领导者用 MTP 和 OKR 的方法论来赢取大成就，这也是 MTP 的最终目标所在。

4.2.3　数学分析决策模型在 MTP 与 OKR 的应用

对于 OKR 的学员来说，我们应该能认识到 MTP 对于业务的重要性。MTP 也决定了业务的资源的投入与方向。

以海外租车为例，我们如果想将目标市场放在中国出境市场，那么整个中国出境人数差不多是几千万人。而在这个出境市场规模里面，具备中国人开车条件的地区主要是：北美、澳新、欧洲部分、东南亚部分地区。日韩等热门目的地不支持中国驾照。那么影响海外租车业务收入（*Y*）的主要变量有哪些?

通过头脑风暴，不难发现，其主要有价格（*X1*）、促销（*X2*）、广告（*X3*）、竞争状况（*X4*），品牌（*X5* 人 产品易用性（*X6*）、服务（*X7*）、驾照支持（*X8*）等。那么在这些变量中，哪些变量是我们现有的数据? 哪些变量需要定义采集新数据? 哪些变量需要换算成其他量化数据?

我们可以将这些数据以日期为单位形成数据表格，并将数据导入工具 Minitab，以验证变量与 *Y* 的相关性。也许我们将得到收入 *Y* 与 *X* 系列的线性回归方程：*Y=B0+*

B1X1+ B2X2+…+B7X7。在得到线性回归方程后，我们分别求得 *Y* 与 *X1* 至 *X7* 的线性相关性情况，以验证哪些变量可以进行进一步的分析。

通过分析，我们将得出中间的强线性相关性变量，并可以将线性强相关性变量定义为 O。通过上述分析，我们从业务层面得到一系列的 O1，O2，O3，O4，O5，O6，O7。这也就解决了 O 从哪里来，以及怎么定义 O 的问题。当然很多情况下，我们可以从经验来确定 O。但一个 OKR 通常需要几个月来推进。如果 O 本身定义的决策精准度不高，这会影响整个业务的推进效果。所以，由 MTP 的目标驱动，来进行整个业务的全面分析，得到核心影响收入 *Y* 的关键变量。将关键变量作为 OKR 的 O，将提升 O 定义的判断有效性。

在 2016 年双周 OKR 检查的过程中，我经常希望大家来验证说明 O 的合理性。原因在于，在产品经理与开发经理的技能要求中，对于目标的判断与决策是其核心技能。

我们在整个地上交通的 OKR 推动过程中，主要以收入经典模型来指导整体的工作。这个在地上交通（火车、汽车、专车、租车）各业务线并不陌生。收入 *Y=X1*(流量)×*X2*（转化率）×[*X3*（单均毛利）-*X4*（单均成本）]。围绕这个经典公式，市场部门主要负责流量目标，产品、技术部门负责转化率目标（面向用户的指标，面向供应链的指标，面向服务的指标），商务部门负责毛利指标。

继续以海外租车为例，从转化率目标 *Y1* 的角度，看影响转化率相关的 *X* 变量的关系。我们一般通过几个维度的数据来推测 *X* 与 *Y1* 的相关性。

1．定价因素对于转化率的影响（*X1*）

2．竞争对手因素对于转化率的影响（*X2*）

3．交互体验因素（费力度）对于转化率的影响（*X3*）

4．不同渠道用户对象对于转化率的影响（*X4*）

针对 1（定价因素）对于转化率的影响，价格因素会影响用户的需求的变化。提升与降低卖价，本身会引发购买意愿的强弱转变。因此，我们可以测试设定不同车型的价格区间，来记录对应价格区间的转化率。这样我们就有一组(*X1*，*Y*)的数据。

针对 2（竞争对手）对于转化率的影响，竞争对手的价格、服务、品牌等因素会

影响转化率的变化。具体原因可以通过调研进一步论证真实性。比如用户会同时在 A/B/C 三个网站间访问，最终用户可能在这三个网站的任何一个来下单。那么，我们要验证，三个网站之间哪个网站会得到更多的订单，以及竞品价格、服务等因素与转化率的影响。研究方法同 1。

针对 3、4 的分析与研究不再赘述。从研究方法来看，我们在日常工作中都用到过 AB Testing、问卷调研、现有数据（流量，订单，用户），这些方法都是为了找到研究对象 *X* 与目标值 *Y* 的相关性。同时通过改进来验证，策略是否有效。

对于 OK 组而言，一般将例如 1、2、3、4 的任何一个目标设置为 O，针对这个目标的相关性进行研究以及验证为 KR。对于部门负责人而言，他们 OKR 的 O 一般为流量、转化率等大的与收入密切的相关性变量。对于总经理而言，他们的 OKR 一般为收入为主的相关性变量。

对于经典收入模型，虽然好用，但是也会有一些缺陷。收入 *Y*=*X1*(流量)×*X2*（转化率）×[*X3*（单均毛利）-*X4*（单均成本）]。针对成熟的业务体系，收入模型是有效的。但这个收入模型没能充分地驱动业务体系如何响应市场的趋势性变化。比如，我们将全部门的资源全投入收入模型公式。过了一年发现，虽然收入的表现是好的，但是产品结构发生了变化。比如汽车租赁，无人驾驶汽车租赁，汽车票与小车城际业务。

所以，在规模化收入基础上，如何保持业务创新与场景创新也至关重要。比如经典收入模型无法说明如何能从汽车票业务中产生船票业务，也无法说明如何从汽车票业务中产生景区直通车业务。

业务创新与场景创新，本质上是从需求的角度对于行业的再认知的过程，重新再回归 MTP 本身的相关性，再来一遍业务的推测判断与业务的认证。

4.3　中层管理者如何看待 OKR

本节先从产品总监的视角介绍了产品团队应用 OKR 的心得、Project 运营方式和 OKR 运营方式的对比，以及 OKR 与 KPI 的对比。然后从技术总监的视角介绍了在实践 OK 制和 OKR 后，不同级别技术人的不同行为表现水准。

4.3.1 OKR 与 Project

我个人一直对于 Google 有比较好的印象，所以我也是早有耳闻 OKR 的体系，可惜一直没有实践的机会。2016 年 Bruce 在车船 BU 推行了 OKR 体系，与原来体系相对比之后，有了些心得，所以写了以下内容和大家分享。

在用车体系下，一直采用的是 Roadmap + Project 的方式。每年我们会定下大的策略方向，然后在此基础上，每个季度会制订个 Roadmap，然后拆解为每个月每个周的 Project，最后执行上线。其实这么运营也没有什么大的问题，好的方面在于如果我制订的 Roadmap 是没有问题的话，大家的目标是很一致的，基本不会扯皮，所有的力气都可以用在刀刃上，没有浪费；不好的方面在于如果我考虑的方向出现了偏差，就会将整个团队带到沟里，比如说包车产品，我们就进行了非常多的不成功的尝试。另一个不好的方面就是产品经理和技术经理的自我思考、自我分析的能力会相对来说被限制了，比如我们的产品处于什么状态？是从 0 到 1，还是 1 到 *N* 的阶段呢？关于某个问题，我们是否还有更好的解决方式呢？产品经理就会比较欠缺这些感觉。

在车船 BU 后，我们开始采用 OKR 体系。对比之后，我个人最大的感觉就是 OKR 更多偏向目标管理，而 Project 更多适合过程管理。怎么说呢？对于 Project 来说，我们要做什么事情，其实是已经比较明确的，尤其我作为技术人员出身，对于 Project 的可行性和实行成本也很容易预估出来，那么剩下来的无非是执行力和技术实现速度的问题。

对于 OKR 来说，则需要每一个产品经理具备自我思考、自我分析的能力，因为要解决的问题不止一个，能解决某个问题的办法也绝对不止一个。所以如何定义清楚合适的 O，其实是一件很关键的事情，因为 O 定义了现阶段该产品需要解决的核心问题，所以我们经常会为着这个争吵，我觉得这也是比较正常的事情。然后就是 KR 的设定，条条大路通罗马，对于 KR 的选择和评估其实也很体现一个产品经理对于产品和业务的理解以及个人的能力。而通过产品会上关于 OKR 的讨论，大家也比较容易了解目前各自关注的重点，以及业务的进展。

另外一个就是角色转变，在 OKR 体系下，我个人更多会从指挥官的角色逐步转化为参谋长、指导员的角色，更多鼓励产品经理对于自己的产品多尝试、多测试。这样，主导权和节奏其实是由产品经理来把握，自然积极的主动性就会高起来，能够比较有效地解决了 Roadmap + Project 的一些问题。而 OKR 的另外一个好处在于，OKR

这种目标管理的方式将大家的共识比较清晰地定义了出来，便于大家沟通和理解。当然，更大的好处是，我会比较轻松，哈哈！

大道至简，关键还是实行 OKR 的人，人对了，OKR 也就对了。2017 年是非常具有挑战的一年，也将会是硕果累累的一年，拥抱变化，拥抱 OKR。

4.3.2　OKR 在租车团队的实践

2016 年随着部门的合并调整，我们对产品运营的方法也采用了 OKR 的思想管理，收获了不少心得，也遇到了不少困难。

OKR 与 KPI 的本质区别应该在于，OKR 管理的是过程，强调驱动人员的主观能动性，强调上级主管的授权和放权。而 KPI 更多的是强调结果，但是对人员及过程的管理会很弱。

我理解的 OKR，首先应该是具有战略的指导作用。OKR 的管理方式要求团队每个周期都要确定一套 OKR，而且数量也应该有一定的限制，我们一般按照 3 个 O，每个 O 由 3 个 KR 来制订。这意味着它确立了团队在这一个周期的核心战略，避免了做那些与核心战略无关的事，提高整个团队做事的效率。我们之前很多时候就是抓不住阶段性的核心战略需要做的事情，导致方向一开始就错了，做了很多事情，但是可能并不能带来很大的帮忙。方向错了，做的事情越多，可能损耗也越大，对于员工来说，这一点也很重要，有很好的主动性的员工本来就很少，而具有主动性同时能很好地规划和执行自己工作的员工更是少之又少，我们花时间好好地制订出一份 OKR，不管对于员工还是团队，都是有百利而无一害的。

其次，OKR 的制订要有挑战性。在制订 OKR 的过程中，我们要求制订的目标有挑战性，也就是要有野心。通过给予团队和 OKR 执行者适当的压力来释放团队的潜力，往往结果都比较令人满意，有可能我们带来了意想不到的增长，也有可能我们并没有预期的增长，但是团队的管理方法和产品迭代过程中的失败教训会让团队成员快速成长。

再次，OKR 会释放团队成员，特别是 OK 组的产品经理和技术经理这两个角色的主观能动性。我们 OKR 制订的过程是团队协商的过程，团队成员通过对数据的分析和理解，共同来识别我们的核心内容，共同识别出我们达成核心目标所需要做的核

心事情。

最后，我们要求 OKR 对团队的成员都是公开的，提高整个参与成员的积极性，同时也减少了沟通成本，大家都知道我们要干什么、怎么干、谁来干、什么时间干，这样透明公开的方式也更容易让团队形成合力，更容易增强团队的凝聚力。

同时，为了防止各个小的 OKR 团队在执行过程中偏离整体大的 OKR，我们也会定期组织 OKR Review，从中间发现我们执行过程的盲点，纠正我们执行过程中的偏差。团队在具体实施半年之后，我们使用 OKR 的方法完成了一系列的产品迭代工作，通过对数据的持续跟进，识别出当前的问题，制订合适的产品及运营策略，使海外产品的订单也得到了大的突破。

4.3.3 一个“演员”的自我修养

2016 年 5 月用车 SBU 和汽车票 SBU 整合，同年 8 月和去哪儿车车整合，同年 12 月收购唐人接进行整合……在着整合的一年中，给我带来最大的收获就是 OK 制运营思路和 OKR 管理方法论。

记得同事打过一个生动的比方：“如果 O 和 K 是导演和演员，那么 OKR 就是剧本，它描述了下一部电影要拍什么，主要的目标是什么，怎么实现这个目标，难点在哪里。”我作为用车演员们的梦想导师，我心目中的演员评级是这样的。

跑龙套

1．前期【制订 OKR】

都是导演 O 的活——事不关己。

参加原型设计，过流程——一言不发。

拿着 UED 和钉钉的任务开始数据库设计，业务架构设计——关注技术。

2．拍摄期【执行 OKR】

开发中遇见业务问题、产品交互问题，找导演 O 沟通——及时沟通。

安排钉钉开发任务，盯着项目开发——按时交付。

1．后期【回顾 OKR】

参加 OKR 两周评审，导演 O 自我总结、自我表彰、自我批判大会——事不关己。

2．点评：拿着死工资呗，这辈子跑龙套的命。

男主角

1．前期【制订 OKR】

和 O 一起制订剧本，量化每个度量，明确哪些 KR 先干，用核心数据说话——事事关心。

参加原型设计，过流程，思考细节，对导演 O 提出挑战，想清楚再撸起袖子一起干——积极参与。

不用钉钉任务，业务设计考虑周全，扩展性强，任务技术风险点提前考虑——事半功倍。

2．拍摄期【执行 OKR】

参加每天站会，随时沟通，不断加戏【技术埋点，监控之类】——积极主动。

按时交付，注重授权，团队建设——水到渠成。

1．后期【回顾 OKR】

监控运营数据，用数据说话，给导演 O 拍续集的信心——继往开来。

参加 OKR 两周评审，和导演 O 一荣俱荣、一辱俱辱——同生共死。

2．点评：中规中矩，独挡一面，这辈子戏是不愁了。

明星

1．前期【制订 OKR】

帮助导演 O 制订剧本，确认目标，量化衡量度量，用核心数据说话，制订 KRs 工作的优先级——激情共创。

确定原型设计、运营流程、KRs 的细节，对 O 进行补位和指导，演出的必是精

品——精益求精。

确定业务设计、系统架构、扩展性、项目风险，经得起业务发展的考验——未雨绸缪。

2．拍摄期【执行 OKR】

主导站会，关注上、下、平级，以及跨组沟通，关注每位同事的开发进度和质量——运筹帷幄。

保证提前交付、监控埋点、接口速度、慢查询、服务性能——无可挑剔。

3．后期【回顾 OKR】

定期对作品进行思考，对行业竞品进行分析，和导演 O 预测未来的趋势——知己知彼。

OK 组负责的领地很繁荣，在短时间内做到了极致的水平，迎接下一个挑战——百战百捷。

4．点评：韩信带兵，多多益善。

一个“演员”的自我修养

- 用心【激情，责任】
- 技术【项目，架构，核心代码】
- 情商【沟通，自我认识，学习能力】
- 思考【深度思考】

4.4 一线经理的 OKR 实战

本节从产品、技术、测试经理的视角介绍了一线实战心得，包括 OK 制和 OKR 如何让产品经理和程序人员消除对立变得齐心合力高效合作，如何运用 OKR 完成 Impossible Mission、OKR 和 SMART 的区别，通过 OKR 激活团队让开发和测试人员变得更有目标和成就。

4.4.1　OK 制是“两个人”的事，OKR 是“一群人”的事

内容简介：分享作为职场新人，从初步接触到熟悉适应 OK 制团队合作的经历，体会到 OK 制对于帮助产品经理和技术人员建立起高效合作的重要方式。在实践 OKR 的过程中，也逐步体会到 OKR 作为一项管理工具和工作理念，在跨团队甚至是跨组织合作中起到非常重要的作用。本文将会以一些案例及个人感受，分享这段时间来的所思所想，希望能对大家有所启发。

最早了解到汽车票团队，是在福泉路的一个不足 50 平米的小房间里，一半隔出 12 个工位，另一半放张大圆桌，也算是个会议室。而初创团队风格，也如这个每天沐浴在阳光的小屋一样，充满了朝气。其中最有意思的，还是团队实行的“OK 制”。

当别人每每谈起“产品汪”和“程序猿”互掐的段子时，最初很难懂其中的点，直到被好多前辈语重心长的教导后，才能领悟到一二。究其原因，莫过于“OK 制”有效地把产品经理和技术人员绑在一起，将两者“天然”的对立关系做到了很好的统一。

相信大多产品经理，要开发资源是其头等大事，天天磨，日日催。而对于开发经理，更多的抱怨则在于对未思考完全的需求不停地返工，甚至对于开发后的产品毫无成就感。毕竟，没有产品思维的技术人员不是好技术人员，而不懂技术的产品经理，还是有机会成为好产品经理的。所以“OK 制”除了促进技术人员和产品经理的沟通，我觉得更大的意义在于双方能建立起长期稳定的默契，能从各自专业的视角提供更好的建议。就拿老板们经常做的种田比喻，我觉得大多团队实行的是老板将地租给佃户（产品经理），佃户通过请教行家（技术人员）指导完成耕种的方式运作。而“OK 制”则是把这块地直接给到佃户和行家，地里的收成除去上缴部分，两者都可以分享到收获的果实。

在实践中，借助“OK 制”个人也学习到很多技术上的行话，什么“解耦”、“重构”，虽然并无什么大用，但至少和技术人员有了沟通基础，更好地理解了为什么总共两句话的需求，在技术眼中需要两周的开发量。比如曾对于模拟出票，概念里无非就是“输入出发、到达、日期后，选个具体班次，然后填下姓名手机号，点下去支付，然后付完钱把取票号回填”，但对于系统的实现，牵扯到代理系统、账号系统、数据抓取系统、扣位系统、支付系统、回填系统、异常处理系统等。在“OK 制”的团队，真正做到了一个岗位学到两份技能。

随着业务和团队的逐步壮大，越来越多的项目需要多个 OK 组的共同参与。一个良好的管理机制“OKR”被引入，进而指导日常工作。比如出票时效和出票成功率，客观上存在一定的互斥，即订单出票成功的概率与给予出票的时间成正相关。因此，对出票系统而言，当然是出票成功率优先于出票时效；反之，对用户服务而言，出票时效意味着用户体验和品牌服务的承诺，超过给定时效的订单完全没有继续尝试出票的必要。

类似这样的“冲突”，双方都各自认为是“正义”的一方，除互相妥协外，似乎很难找到较好的处理方式。但如果在“为用户提供及时稳定的出票服务”的 OKR 下，为双方的争议点找到了一个共同的评判标准，例如出票系统，可以用出票失败可能引起用户投诉的比例为自己争取，而服务组则可以用出票时效对用户满意度的提升进行证明。虽然两者在用不同的“武器”捍卫自己，但基于这个统一的目标，量化出如客户价值等度量工具，便能将双方的博弈直接进行对比。除了提供统一的度量，更大的意义在于识别一些看似有意义但实际对目标达成无贡献的因素，同时拓展各组的思维。不要只纠结于出票成功率、出票时效，或者及时稳定的出票服务，是否还可以包括出票系统自身优化、用户的预期管理、延迟出票提醒、备选班次选择等新的技术或者产品形态。

当然，我对于“OKR”的认知还是停留在自我消化过程中，虽然不太认可存在即合理的观点，但始于 Google 等巨头的 OKR 管理方式肯定会有其更深远的效果。唯有不断增加积累，希望能在不远的未来，在一些看似微不足道的细节启发下，迎来量变引来的质变，找到突然的“大彻大悟”。

4.4.2 完成不可能目标的可行方案

曾经流行的一个个人管理方法叫 GTD，实际上就是任务列表法。它要求事无巨细，把所有想干的事情都作为一个个“项目”，每个项目又分为若干个“动作”，每个动作都有自己的规定完成时间、地点和预期难度。当你做事的时候，你要处理的并不是项目，而是动作。这个做法带来了条理，带来了高效率。

20 世纪 80 年代，通用电器把这个方法发挥到了极致，变成一套叫“SMART”的目标系统。这套系统也就演化成另外一个管理体系——KPI。个人有主见，上级有监督，做事有节奏。通用公司通过这套体系，八年股票翻了三倍。

但时间久了，我发现有些部门，虽然用了，效果没见，反而利润下降了。问题在于，他们过分追求“完成”这个动作，而放弃了原有的初衷。

本来火车纵穿日本需要花费 20 个小时，日本政府为刺激经济增长，提出了建造更快的火车。工程师计算，以当时技术能力，火车时速大约提高 65 英里/小时，但政府希望火车时速能提高到至少 120 英里/小时才有可能让经济发生质变，但 120 英里/小时时速的火车当时根本不存在。

工程师：75 英里/小时也可以努力。

政府：给我 120 英里/小时不可行的理由。

工程师：比如转弯的时候火车肯定翻车。

政府：为什么非得转弯？建隧道！

就这样，政府先给一个看似不可能的目标，然后一点点按照这个目标去解决，最终完成了新干线。日本搞这个目标，显然不符合 SMART 原则，看上去不可行，但反而引领整个铁路系统办成了一件了不起的事业。他把这种做事方法称为“Stretch Goals”。

做事不能光考虑可见的具体目标，更要有那种现在根本无法清晰描述的长远目标，不管怎么说先定下来，往这个方向尝试。这个管理实践方式，就是我们火车票部门从技术到产品服务借用的 OKR 管理方式。

OKR 是高效与自由相对平衡的一种管理机制，核心就是通过统一的目标驱动、自由可变的方法、高效灵活的应变手段，在我们日常工作学习中都得到充分利用。

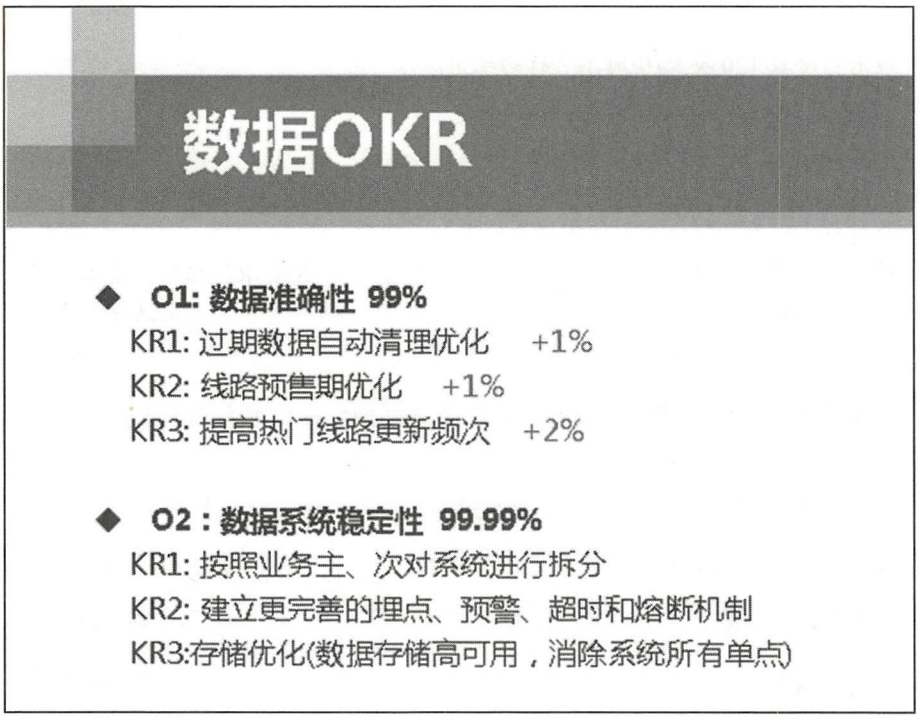
它与其说是一套惯例制度，我却理解为一套思想体系。自觉，这套思想体系，在我们技术组或在我们出票组发挥得最深刻、最完善。众所周知，出票系统是所有生产系统中最不确定的，作为其产品经理，不但要配合开发人员，还要研究将来的可能性、安全性，执行力和管理能力都要靠一套完善的思想作支撑，遇到不确定，都要自由运用有限的手段解决。

总的一句话：目标重要，手段不重要。这句话，非常深刻，就是运用了 OKR 精神。我们在 2016 年完美回收了去年年底汇报时放出的目标，完成了看似不可能的出票成功率 XX%指标（当时喷了句这不现实，现在想想很惭愧）。最重要的是，我们今年还可以抽调资源，支持其他项目组的建设，年中还能集体出游一次，在过去，那是想都不敢想的事。

在此，感谢 OKR 的实践。我们将来会通过不断尝试，争取再创更多不可能的成就。

4.4.3 OKR 在汽车票数据系统的实践

在 2016 年年初，汽车票数据系统制订了 2 个重要的 OKR，如下图所示。



汽车票数据系统的主要职责是提供稳定可靠的数据服务给各个汽车产品线，包括携程 APP、H5、铁友、汽车票独立版以及分销渠道等。在数据系统的上游有两个大的子数据系统，分别是直连数据系统和模拟数据系统，由两个子数据系统分别对接不同的供应商，完成数据的采集，汇总到数据系统。

当时存在一个比较严重的问题：汽车票前端展示的数据和供应商网站的数据很多时候会有差异，比如供应商已经下架的班次我们还在卖，或者供应商新增了班次，我们却没有及时更新。由此会产生很多问题，尤其是地推方面每天都会报告很多班次和线路的问题。

如何尽可能地和供应商的数据保持一致，是迫切需要解决的问题，所以产生了第一个 O——数据准确性（99%）。我们首先建立了数据中心和子系统（直连/模拟）数据接口对比工具，数据对比显示当时的数据一致率在 95%左右，借助对比结果和对系统进行仔细分析后，制订了 3 个 KR：过期数据自动清理优化，线路预售期优化，提高热门线路更新频次。在这个 OKR 完成之后，数据系统和供应商的数据一致率已经非常接近 99%了。

再来说说第二个 O——数据系统稳定性。数据的重要性不言而喻，一旦数据系统出现抖动，那么所有的汽车票产品都会受到影响，造成订单量下跌。所以，保证数据服务的稳定也是数据系统的重要职责。因为汽车票业务的迅猛发展，汽车票数据系统在此之前也是疲于应对业务需求的变化，系统本身的健壮性并不理想，存在不少风险因素。而业务体量发展得越大，对数据稳定性的要求也越高，所以我们把稳定性的目标定为 99.99%，并基于对现有系统的风险分析和以往故障的总结，找出了目标的 KRs。完成这个 OKR 的结果也是令人满意的，在过去的一年时间里，汽车票数据系统仅有 10 分钟左右的故障时间，由代码发布所产生的故障次数更是降至 0 次。

心得体会：很多时候，技术人员都是忙于应对各种各样的 CR，尤其是被产品需求的反复修改折腾得死去活来，但到总结的时候又不知道自己到底做了些什么。我觉得 OKR 管理方式不仅可以帮助技术人员在一段时期内有明确的目标，变被动为主动，并且在完成 OKR 之后获得更多的成就感。

4.4.4　从坚强后盾到挑战创新——测试团队 OKR 实践

2016 年 8 月我到火车票 BU 接手测试团队管理，测试团队在 8 月的年中颁奖会得到了坚强后盾奖，同时有一半的团队成员因为几年的点点点型重复测试导致职业倦怠和没有发展而提出离职。火车票 BU 这两年的 OKR 实践覆盖了产品和开发团队，没有覆盖到测试团队。我的第一板斧就是给测试团队导入 OKR，因为 OKR 强调挑战精神，正好适合激活团队。

传统的目标管理不管是 KPI 工具还是 SMART 原则都强调目标可达成，不达成便是不合格的，甚至直接挂钩绩效考核影响奖金。OKR 强调目标有挑战，关键结果有逻辑和创新，通过创新的方案和行动来冲刺挑战性的目标。如果 OKR 执行尽了最大的努力，目标最终达成了 60~70%，这是挑战较好的设定。如果目标达成了 100%，说

明目标设定时的进取心还不够强，无法探索出我们体内的最高潜能，就像学校考试大家都得 100 分，试卷设定对学生缺乏区分度一样。

OKR 这种精神促使每个测试同事开始深度思考除了按现有测试方式保障产品质量，可以有什么样的小目标来提高工作效率和质量、改善工作方式、提升专业水平，大家开始变得有活力。经过几个月的实践，团队形成了 OKR 一季一设定、两周一回顾的目标管理机制。通过导入 OKR 促动的挑战和创新，团队从多年的点点点型测试跨出了突破性的第一步——实现了携程 APP 火车票主流程 API 的自动化测试，节省了每个版本的回归成本，并且从手工造单转向自动造单，节省了测试数据的准备时间。

测试团队的 OKR 实践在透明化方面有个特色之处，每个人的 OKR 都用卡片展示在显示器上方。第一，OKR Owner 每天在办公桌时时能看到自己的 OKR，高度聚焦，超越日三省乎己；第二，方便合作伙伴了解自己的 OKR，更好的协作；第三，公示个人 OKR 可以带来同行压力，也是奋进的动力。

测试团队的管理机制除了目标管理机制（OKR 迭代）成型，还健全了绩效奖惩管理（设定了明确的绩效晋升方案和质量罚款规则）、专业发展管理（塑造了每月阅读和分享的氛围）和健康开心管理（形成了每周运动和每月活动的文化）机制。测试团队从之前的保守、倦怠、沉闷、离心变成有理想、有底线、有成长、有活力的四有新团队。2017 年，我们一起在挑战创新的路上继续前行！

NO.3
技术篇

打造高效的敏捷团队，不仅和组织架构、研发流程、团队氛围有关，还需要强大的工具支持，才能切实提升团队的交付能力。在这里，你可以找到覆盖产品生命周期全链路的各种工具，看它们是如何帮助我们提升技术效率的，希望同样可以帮助你。

第 5 章

效率为王，唯快不破

工欲善其事，必先利其器。要想打造一支一流的敏捷研发团队，不仅要养成好的工作方法，设计好的组织架构，获得强大便捷的工具支持也是必不可少的。

我们一直在为研发团队量身打造能覆盖整个产品生命周期的管理工具链条，涵盖了软件开发、测试、发布、运维等各个阶段。通过工具的不断完善与改进，促使交付效率不断提升。

接下来，我将从闪电交付、敏捷运维、沟通协作三个方面，谈谈我们为打造敏捷研发团队，在技术、工具方面所做出的努力。

5.1 闪电交付

产品业务价值的提升，本质上是对软件功能的持续改进，它的关键是要在用户与项目团队（包括客户或者 Product Owner）之间建立紧密的反馈环，要建立这个闭环依赖于团队是否有这样的能力，即通过持续交付新的软件版本，验证新的想法和软件的改动，并能衡量这些改动对收入的影响。

对于很多需要几个月时间才能发布新版本的公司来说，在一天之内发布数次新版本似乎是天方夜谭。然而，只要遵循了持续交付的原则与实践，原来一个月才能发布几次新版本的产品，现在已经能在一个周内发布数次版本，或者更频繁了。这是一种

巨大的竞争优势，而且意味着，在时间和精力方面减少了大量的浪费。

截止 2016 年年底，我们总应用数在 5000 个左右，平均每周约有 3000 次以上的发布需求，平均单产品线每天生产发布数 10 次以上。

5.1.1　持续集成，让团队没有难集成的代码

持续集成（Continuous integration，简称 CI），即团队开发成员经常集成他们的工作，在每一天的时间中，可能会发生一次或者多次集成。而每一次的集成都是通过自动化的构建（包括编译、发布、自动化测试）来验证，从而尽早地发现产品中的错误。

我们研发团队勇于持续交付实践，经过三年多的努力，形成了特有的一套持续集成体系，为研发质量与效率的提升打造了良好的基础设施。

在这三年中，我们技术中心在 CI 上可以做到：

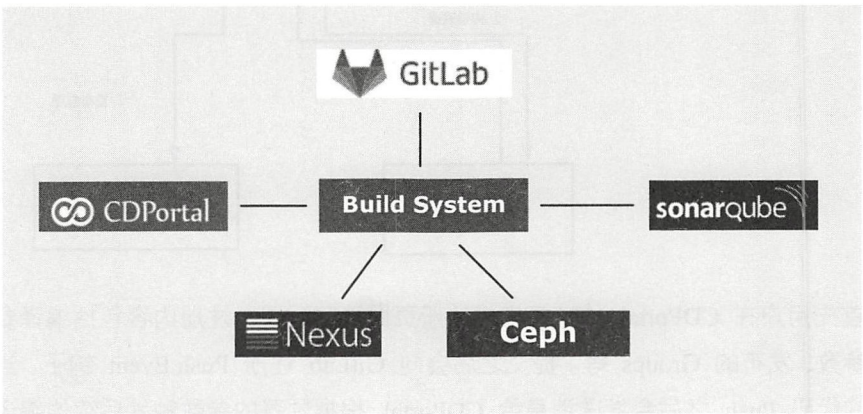
1．建立了集团统一的代码平台。

2．打造了秒级的构建系统。

3．形成了代码自动构建及自动检查机制。

4．软件可自动部署到 Docker 容器。

整个研发 CI 的环境如下图所示。



各服务的简要功能如下：

1．CDPortal：持续交付的入口，也是持续集成的配置中心。

2．Build System：基于 jenkins 自主研发的构建系统。

3．GitLab：代码管理平台，在 GitLab CE 版本上做了二次开发，具备灵活的可扩展功能。

4．Nexus：组件依赖管理系统。

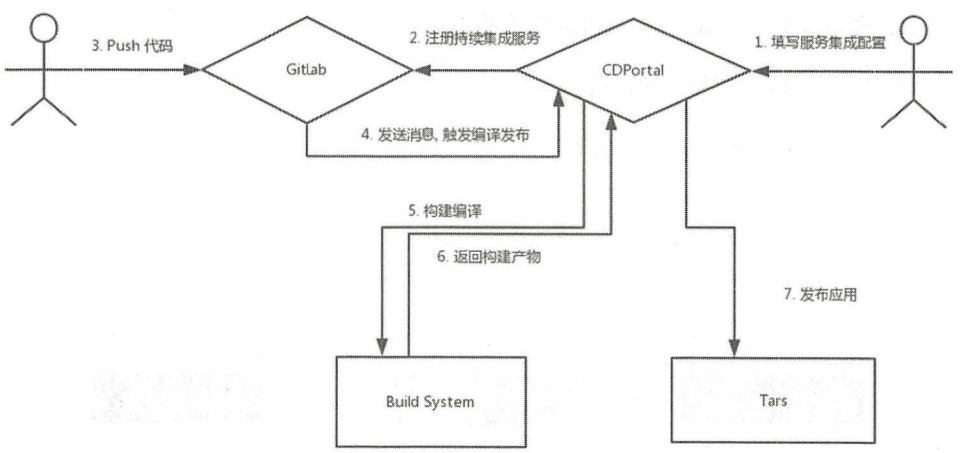
5．Ceph：持续集成的存储系统。

6．sonarqube：静态检查管理系统。

我们精心设计了“CDPortal”、“Build System”和“GitLab”服务，积累了一定的经验或教训，下面分别做详细的说明。

持续交付门户——CDPortal

CDPortal 是我们自主研发的持续交付平台，其中包括编译、发布、环境管理等一整套交付闭环，在持续集成方面，CDPortal 的主要实现流程是这样的。



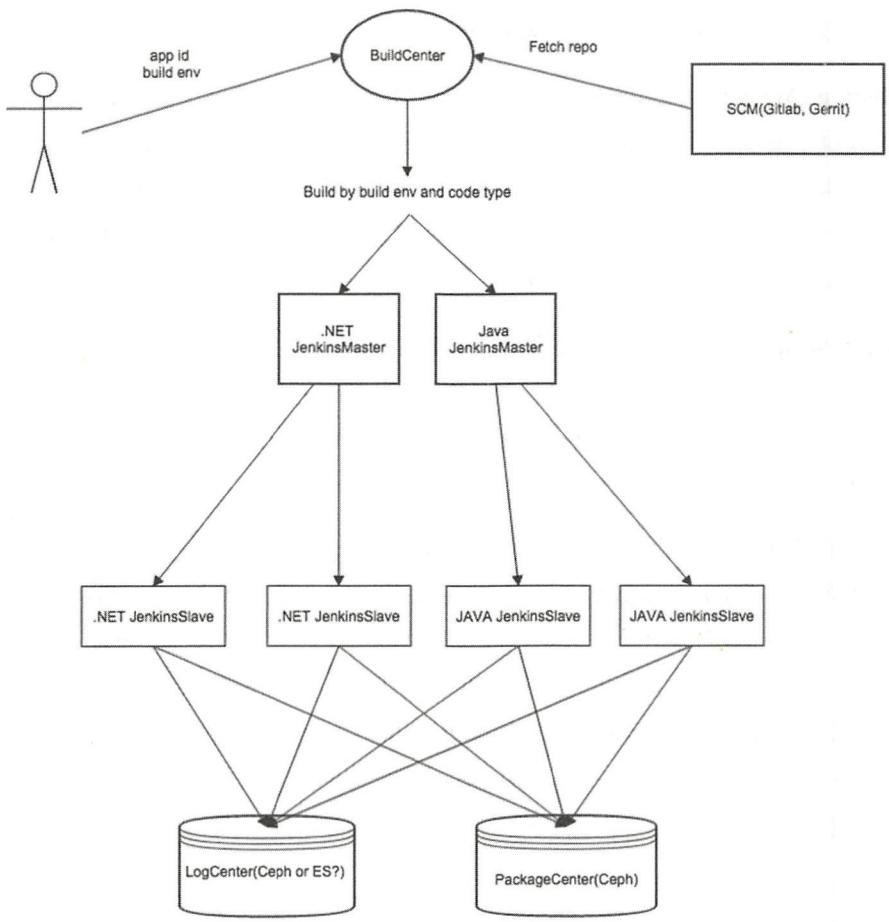
首先用户在 CDPortal 的持续集成注册页面填写配置，注册内容包括编译参数、发布参数、发布的 Groups 等。提交之后会向 GitLab 注册 Push Event 钩子，当用户有新的代码 Push 之后会发送消息给 CDPortal，根据预置的参数触发后续的编译发布操作。每一次触发都会形成一个版本，每个版本的代码包或者 Docker Image 都会被

保存很长一段时间，方便用户进行回滚等操作。对发布周期重要的节点可配置 WebHook，用于连接用户自己的服务，比如在发布 FAT 之后通知自动化测试工具执行自动化测试。

CDPortal 在发布的时候还提供了非常多的工具帮助用户可以更好地管理项目，如在发布周期添加 Sonar 对代码的静态检查。发布过程可以选择添加 JaCoCo 服务查看 Java 代码的覆盖率，测试环境是否集成 Mock 服务器等。

构建——Build System

Build System 支持多种类型的构建操作，构建操作来源于 CDPortal、GitLab、AresPortal 等发布平台，具体架构图如下所示。



用户通过发布平台传递构建参数，构件参数包括源代码，系统通过 SCM 系统拉取源码，将构建请求发给 Jenkins Master 集群，Master 集群选择合适的 Jenkins Slave 运行构建任务，构建运行的产物及日志最后上传至分布式存储系统 Ceph。该项目主要包括如下组件。

BuildPortal：基于 Ruby On Rails 开发的 Web 及 RESTful API，包括 job 管理、build 管理、Jenkins 管理、构建配置管理等相关功能。接入 BuildPortal 的构建类型必须有 2 个以上的 Jenkins Master 保证该类型高可用，每种 Jenkins Master 可接受多种类型的构建。当构建请求发起时，BuildPortal 根据当前 Jenkins Master 的负载情况选择将任务发送给它，若一台 Jenkins Master 不可以用，自动切换至其他可用的 Jenkins Master，并发起报警邮件，BuildPortal 本身也集成定时任务去监测当前 Jenkins 的可用情况。在发送任务之前，BuildPortal 还提供了构建配置管理，配置管理带有版本，方便进行灰度测试。每次构建之后，BuildPortal 也会给予相应的邮件通知提醒。

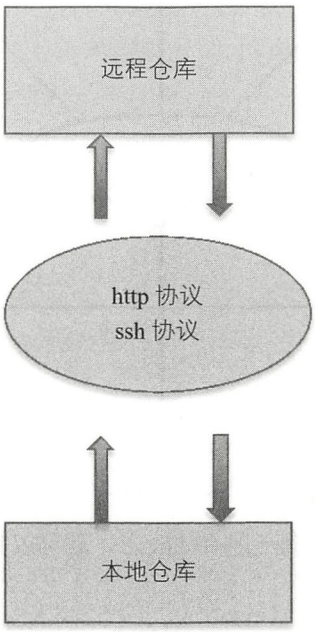
Jenkins Master 与 **Jenkins Slave**：Jenkins Master 主要负责队列调度，本身不负责任何构建操作的执行。Jenkins Master 可以挂上多个 Jenkins Slave，当收到构建任务时，根据任务类型分发给合适的 Jenkins Slave，每种类型至少保证有两台 Jenkins Slave 保证高可用。除此之外，Jenkins Master 还提供对模板 Job 配置的更新操作，Jenkins Slave 是基于 Docker 的容器，便于管理和扩容。

BuildScript：BuildScript 是负责具体构建过程的脚本文件，其中包括多种类型的构建逻辑。

代码仓库——GitLab

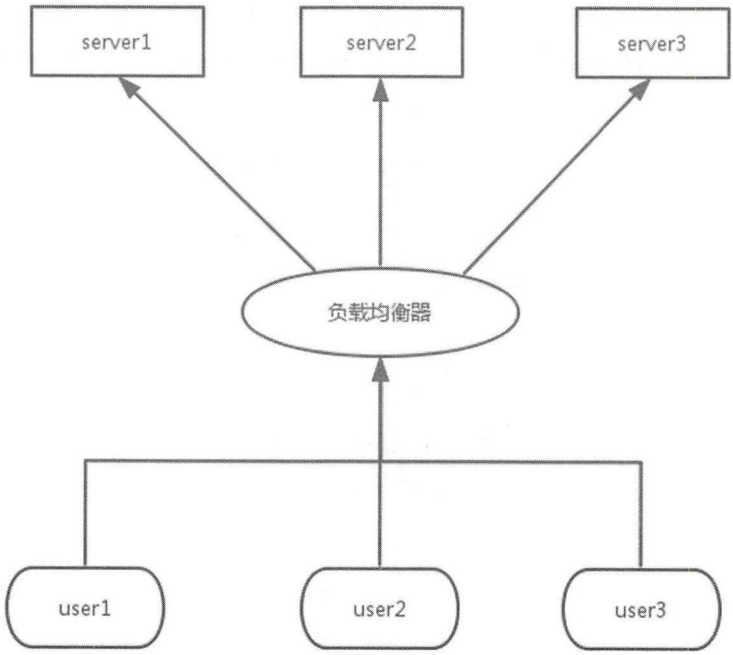
我们在 2014 年引入 Gerrit 作为代码管理和 Code Review 的平台，系统良好地运行了两年多。随着 Git 仓库数量的增多，以及每个 Git 仓库的增大，从 2016 年 2 月开始，Gerrit 的 http 服务隔几个月会出现故障，季度可用性一度跌至 99.55%。为了保证高可用性，我们分析了 Gerrit 的具体实现，了解到 Gerrit 的横向扩展能力比较弱。同时，考虑到 Gerrit 的 UI 和 API 都不及 GitLab，因此，我们果断决定用 GitLab 替代 Gerrti，使 GitLab 成为我们统一的代码平台。如果要保证 GitLab 高达 99.99% 的可用性，必须改变单服务器的模式。下面我们把 Sharding 的实现原理和具体实现做详细说明。

使用 Git 作为版本管理工具，进行项目的合作开发，需要借助远程仓库，供项目的不同成员统一存储代码及变更。而基于本地仓库和远程仓库之间的同步通常情况，会使用 ssh 和 http 协议进行，如下图所示。

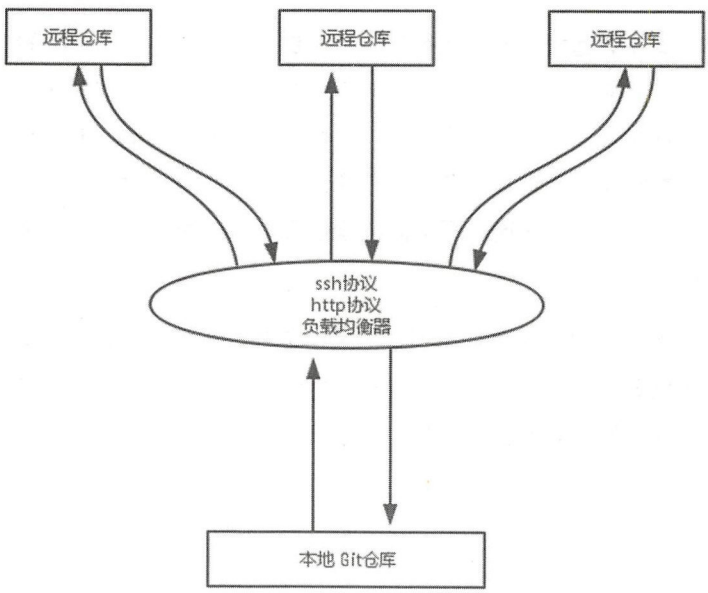


在这种模式下，远程仓库将运行在一台机器上，而本地仓库会随着开发人员的增多而不断增多。这种模式对于开发人数并不是很多的中小团队非常适用，而对于开发人数上千、上万的公司，远程仓库使用一台机器的弊端将会彻底暴露出来。此时因开发人数很多，本地仓库会频繁地和远程仓库进行交互，从而对服务端机器的配置和性能有很高的要求，并且万一这台服务器出现故障，将会影响所有的开发人员，容错方面会非常糟糕。

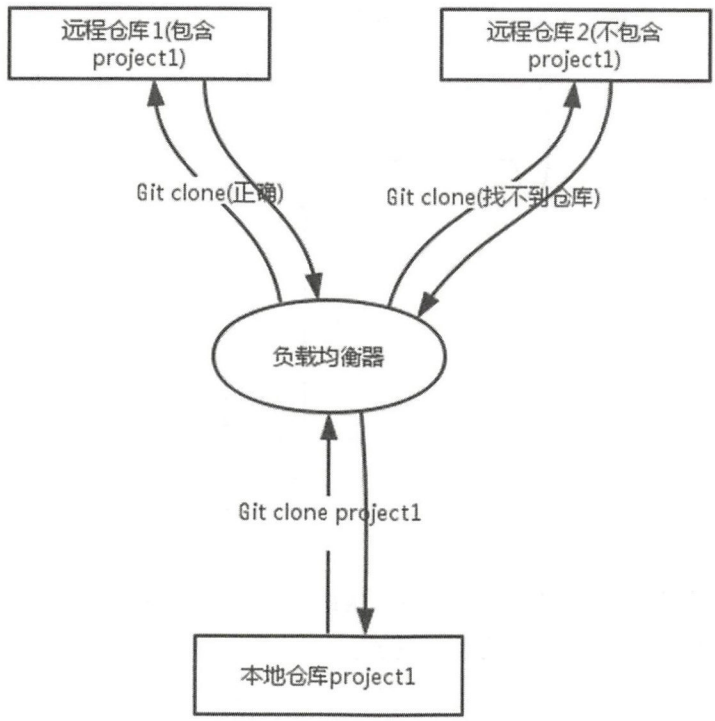
通常一个网站在遇到这种瓶颈时，会采用负载均衡的方式，将服务端的压力由一台分摊到多台机器，从而减轻单台服务端机器的压力，实现由多台性能一般的机器代替一台性能非常高的机器的目的。如下图所示。



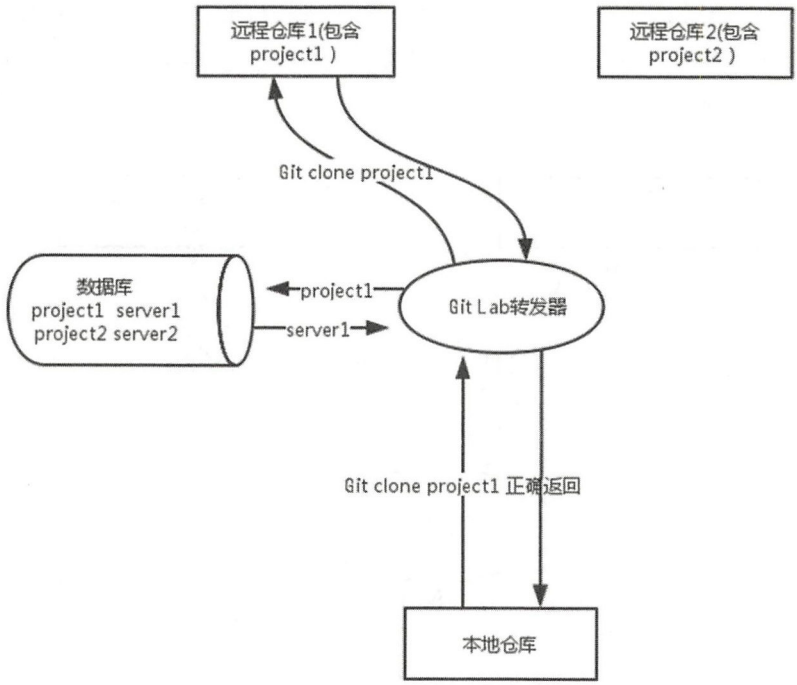
由此启发，如果将负载均衡迁移到 Git 上，就可以完美地解决 Git 单台服务器负载过高的问题，如下图所示。



但是这个方案存在致命的问题，因为 Git 的远程仓库是存储在磁盘上的，所以当使用负载均衡器访问远程仓库时，因为并不知道将要访问的仓库在哪台服务器（server）上，所以可能会将请求转发到并没有此仓库的服务器上，从而访问不到远程仓库。如下图所示。



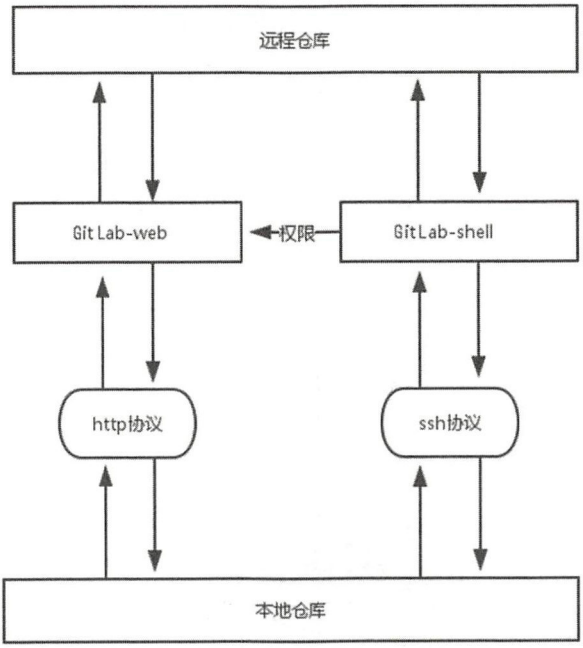
为了解决这个问题，可以将 project 在哪台服务器上记录在数据库中，然后在转发访问请求的时候，读取这些信息并转发到对的服务器上。如下图所示。



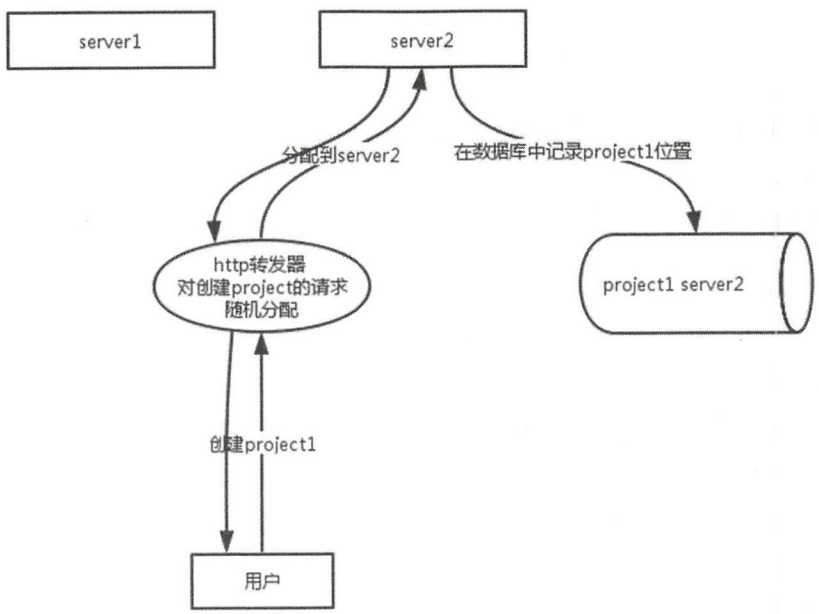
沿着这个思路，就可以在几乎不损耗性能的情况下将单台服务器的压力分摊到多台服务器上，并在容错方面有了很大的提升。

具体实施方式

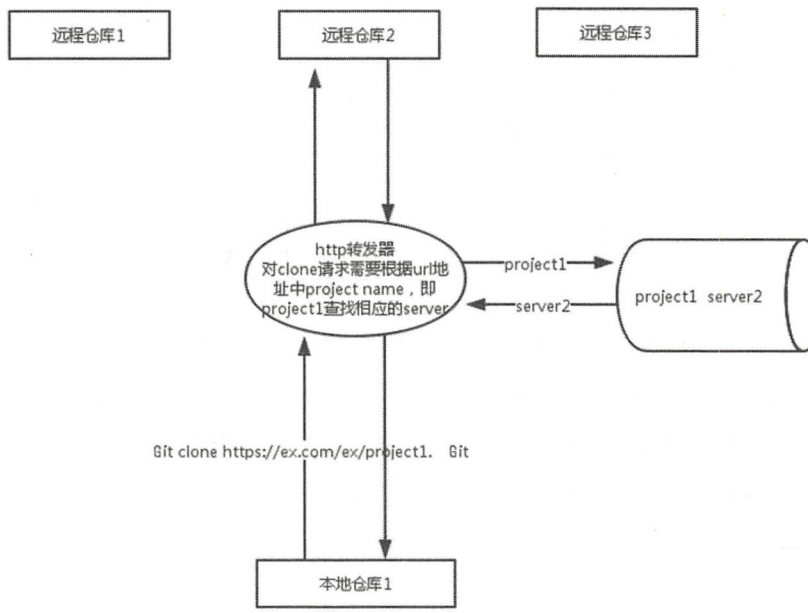
下面以 GitLab 为例，讲述具体是如何实施的。GitLab 是利用 Ruby on Rails 开发的一个开源的版本管理系统，实现一个自托管的 Git 项目仓库，可通过 Web 页面进行访问公开的或私人的项目。下图简单地展示了 GitLab 的工作原理。



因为我们要实现多台服务器供用户访问使用，而不同的项目可能存储在不同的服务器上，所以在创建项目仓库的时候，需要记录项目仓库是在哪台服务器上创建的。本例是采用数据库存储仓库和服务器的对应关系，如下图所示。

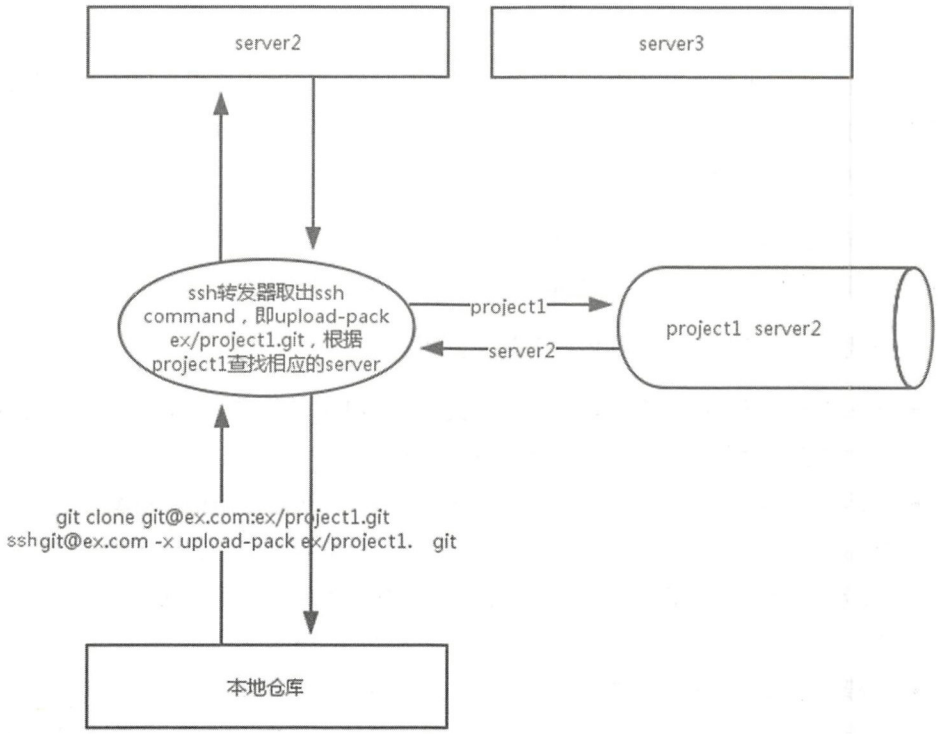


然后当通过 http 协议访问仓库时，比如想 clone project1 到本地，需要运行 Git 命令：`git clone https://ex.com/ex/project1.git`。当运行这条命令时，实际上是 Git 在底层访问了这个 url：`https://ex.com/ex/project1.git/info/refs?service=git-upload- pack`，而项目名称 project1 在 url 中就能获得。当通过分析 url 获得项目名 project1 后，就可以通过数据库查找到 project1 所在的服务器 server2，整个工作流程如下图所示。



当使用 ssh 协议和远程仓库交互，想 clone project1 到本地时，需要运行 Git 命令：`git clone git@ex.com:/ex/project1.git`。而当运行这条命令时，Git 内部实际上执行的命令为：`ssh git@ex.com -x upload-pack ex/ project1.git`。

此时，这条命令的意思是在 git@ex.com 这台服务器上执行 `upload-pack ex/project1.git` 这条指令。所以，要完成 ssh 的转发，需要在转发器中截取到这条指令，然后从指令中取出项目的名称，即 project1，然后在数据库中取出 project1 对应的服务器 server2。此时再由 ssh 转发器向 server2 发送 ssh 请求：`ssh git@server2.com -x upload-pack ex/project1.git`，然后由 ssh 转发器将从 server2 接收到的数据回复给用户，而从用户处接收到的数据则转发给 server2，整个工作流程如下图所示。



http、ssh 转发器的实现可以根据各自的喜好用不同的语言实现，我们使用 nodejs 实现了转发器的编写。当然，sharding 能解决单机模式下设备故障导致服务全不可用的问题。不过，如果要保证各个 sharding 服务都可用，我们还应该对每台 GitLab 做备份。为了节约成本，我们目前把各个 sharding 上的 Git 仓库都备份到另一台设备上，一旦线上的单台设备出故障，可以通过在新设备上恢复数据的方式快速恢复服务。

依据上面的设计，我们花了三个月进行开发与测试，在 2016 年 6 月中旬上线了具有 sharding 功能的 GitLab 代码管理平台。截至 2017 年 7 月，GitLab 季度 HA 都保持在 99.99% 之上，近三个月活跃的 Git 仓库有 6000 个，并且只需几个简单的配置就能添加一台 sharding 服务器。

5.1.2 后台应用持续交付实践

作为国内最大的 OTA 公司，我们为数以亿计的海内外用户提供优质的旅游产品及服务。随着我们业务量迅速增长，业务变化越来越敏捷，对于应用交付的效率也提出了更高的要求。根据统计，截止 2014 年年底我们总应用数在 5000 个左右，平均每周约有 3000 次以上的发布需求。

所以作为整体交付环节中极为重要的一环，应用的部署和发布是提高交付效率的关键。然而我们原来的发布系统 Croller 却成了阻碍交付效率提升的一大瓶颈。

关于我们的火车发布，下面做一下简单介绍。

- 火车发布规定：每天定时安排发布车次，以 pool 为单位安排车厢，在一个 pool 中的应用必须在“同一车次”的“同一个车厢”内做发布。
- 实际发布情况：每个应用在发布前需要“买票”，也就是申请和备案的过程，然后被分配到某个“车次”，与同在一个 pool 且需要发布的其他应用形成一个“车厢”，当到达规定发布时间时，该“车厢”内的所有应用以灰度的方式做发布。
- 该模式的弊端：如果提前准备好了发布，未到达规定发车时间，只能等待，不能发布；如果错过了某个发车时间点，只能等待下一次；如果发布过程中，同一个车厢内有一个应用发布失败，则整个车厢中的应用全部发布失败。

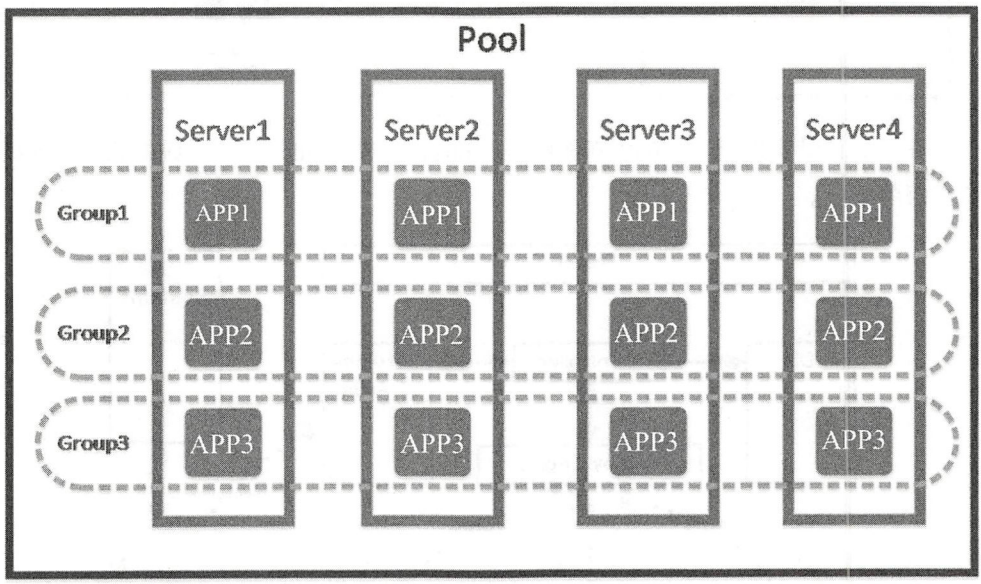
具体来说，我们 Croller 设计的是火车模式发布，包括以下主要面临的核心问题。

- 由于 ASP.NET 的应用占大多数，基本都采用的是 Windows + IIS 的单机多应用的部署模式，应用和应用之间的隔离性较弱，并且由于应用划分的颗粒度比较细，在单机上往往可能同时部署 20~30 个应用，多的甚至达到 60 个，导致大量不同的应用之间共用应用程序池的情况存在，即多个应用运行在同一个进程下，这种情况下任何一个应用的发布都可能影响到其他的关联应用。
- 使用硬件负载均衡设备承载应用的访问入口，以域名为单位隔离。单机上的多个应用程序共享同一个访问入口（同一个域名），所以健康检测也只能实现到服务器级别，无法识别应用级的故障。
- 由于治理系统中的应用信息不统一或不准确，影响监控和排障。

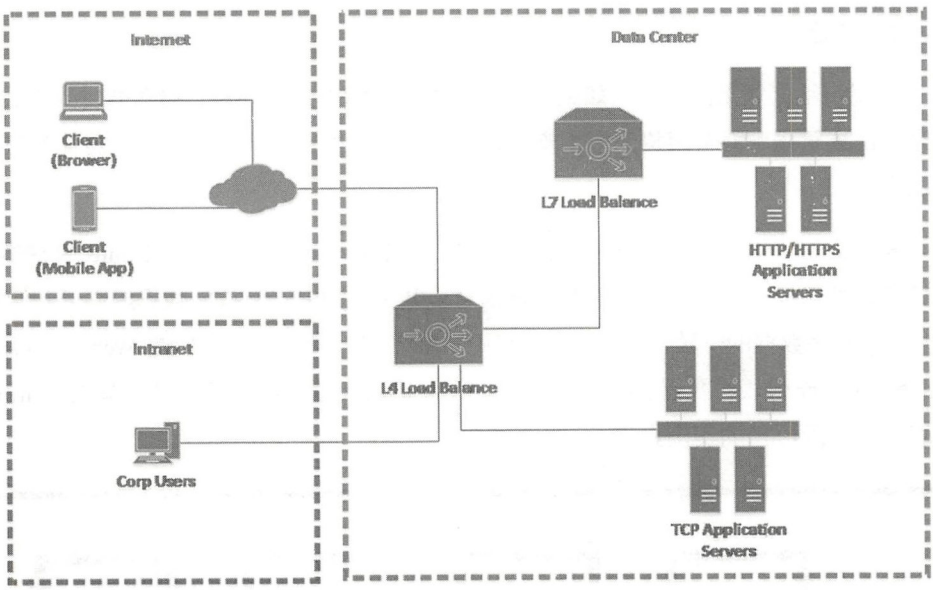
1．破题思路

针对混乱又复杂的情况，如果要想从根本上去解决这些问题，提高交付效率，则必须从配置管理、部署架构上全面支持以应用为最小颗粒度的管理能力。具体有三种解决方案。

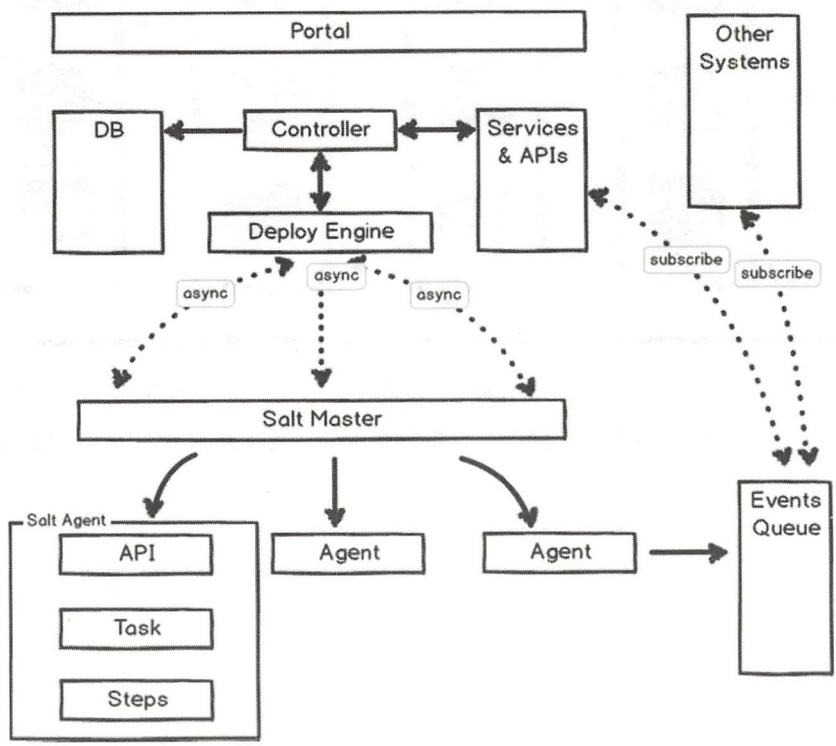
第一，引入 Group 的概念，设计从 APP、服务器、Pool、Group、Route 的完整数据结构模型来描述应用相关的配置部署信息，并由 CMS 作为权威数据源向外提供数据接口，确保数据的一致性和准确性。其中 APP 代表一个应用，可以是 Web、Service、Job 等；Server 代表服务器；Pool 代表部署了相同应用程序的一组服务器集合；Group 代表一组相同应用的实例集合。



第二，引入 7 层负载均衡（SLB），实现应用访问入口的隔离。使每个访问入口（集群）的成员（即应用进程实例）可具备独立的管理能力，实现应用级的健康检测。



第三，设计实现新一代的发布系统——Tars，解决 Croller 发布系统的痛点，支持应用级的发布。



2．落地解题

虽然有了破题思路，但具体实现仍然有很多细节需要考虑，主要包括统一配置（CMS）、弹性路由（SLB）、想发就发（TARS）。

统一配置（CMS）

正如大型传统企业发展初期缺失 ERP 系统一样，互联网公司也需要发展到一定规模才能意识到一个完备的配置信息系统的重要性。互联网公司在整个产品研发和运行生命周期中不断产生大量的系统和工具，例如测试平台、发布平台、监控系统和资源管理工具等。业务产品研发效率和业务系统稳定运行，依赖这些工具平台的高效协同工作，而如果要实现这种高效协同工作，关键就是拥有一个统一的配置信息平台。

不成熟的配置管理往往有以下特征：

- 配置系统之间相对独立和分散，缺少关联关系，并且运维、研发工具、测试生产环境都有各自视角的局部配置管理系统。
- 缺少明确的定义和抽象。例如，不同语言开发的应用对配置的描述方式有很大的差异性；或者对集群、发布节点和访问入口等重要对象的定义很模糊。
- 配置信息不准，依赖手工维护，没有工具和流程约束，要执行者自己来保证操作和配置数据的一致性。
- 配置描述不完整，使得系统架构，比如集群、域名映射等关键环节缺乏严格的配置管理。

我们这种配置管理暴露出很多问题，当监控到服务器资源异常时，例如 CPU、内存出现异常，无法快速查找该异常影响的范围，造成不知道应该联系谁进行排障；而对于非.NET 的应用无法提供发布工具，或只是提供了一些功能有限的发布模块，但由于不同技术使用的发布工具有着很大的差异性，给使用方和开发维护方都带来了极大的不便；当资源和应用之间的关系不清晰，运维无法实现完善的资源计费等重要的管理职能。

要应对这些问题，需要定义统一的配置模型和一致的配置数据，清晰地描述组织、业务、应用、系统架构和资源等要素及互相间的关系。从更高层次设计一套配置管理系统，使得各个维度的配置信息既要专注于自身的领域，又能和其他相关的配置信息维度建立起关联，确保这些工具能以一致的定义来理解配置数据，进行顺畅而有效的

协同工作。于是我们的配置管理系统 CMS 就应运而生了，CMS 的核心目标包括：

- 数据准确（即与实际保持一致）且合规
- 数据关系的查询方便高效
- 数据变动可追溯
- 系统高可用
- 数据模型简洁易懂

CMS 系统演变过程

1．抽象、定义、建立关系、存储数据

对于应用层面运维所涉及的对象进行统一抽象，使得使用不同技术、不同架构的应用体系都能使用一样的模型结构来进行描述。根据我们的应用体系和管理方式，抽象出一套核心的应用配置对象，包括组织、产品线、产品、应用、集群、发布节点、服务器等。

经过与那些不同语言、不同技术架构所开发的应用间的磨合实验，我们验证了这套抽象的配置对象有其普适性，并可以完备地描述我们范围内各种应用的配置状态。只要按照这套配置对象系统对一个应用完成了描述，那么该应用从发布到上线运行再到下线的生命周期内，各种相关工具均能通过获取这些配置状态得到足够的信息进行工作。换句话说，通过这套统一的配置信息数据库，不同开发者、不同阶段、不同功能的平台实现了协同工作。

2．将 CMS 作为一种服务提供出去

由于建立了描述应用体系的核心配置数据库，这必然会有大量用户和工具成为 CMS 的消费者。所以我们希望 CMS 的消费者可以通过网络随时随地获取、维护和管理 CMS。这要求 CMS 能提供完备的 API 和一套简洁直观的管理界面。

3．通过 Portal 和工作流引擎完成配置变更，实现业务逻辑的自动化执行

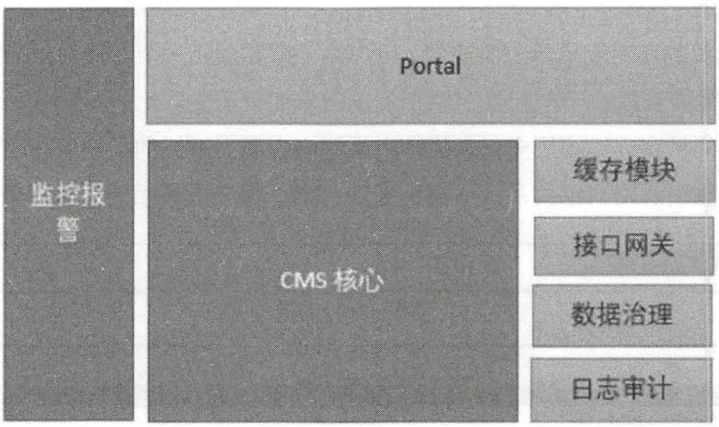
除了建立统一的应用配置模型，还要建立应用配置的生命周期管理，做到生成配置、修改配置以及销毁配置都合规，都经过授权，都有记录可查。

4．搭建一个强壮可靠的配置管理体系

通过更多的子模块帮助搭建配置管理体系来提高稳定性和可用性，实现查错追溯和数据巡检纠错等功能。

CMS 系统架构

CMS 系统在开发过程中遇到和解决了一系列的棘手问题，系统本身的架构也反映了这些方案的设计实施情况。



数据治理

CMS 系统最基本和关键的需求是提供正确的数据，数据必须能真实反映生产环境的配置现状，并且还要符合公司制订的运维规范，不能出现违规配置。例如，我们不允许同一个应用在一台服务器上运行多于一个实例；不允许在一台服务器上运行多于一个的 Java 应用；每台服务器上只能运行同样类型的应用等。

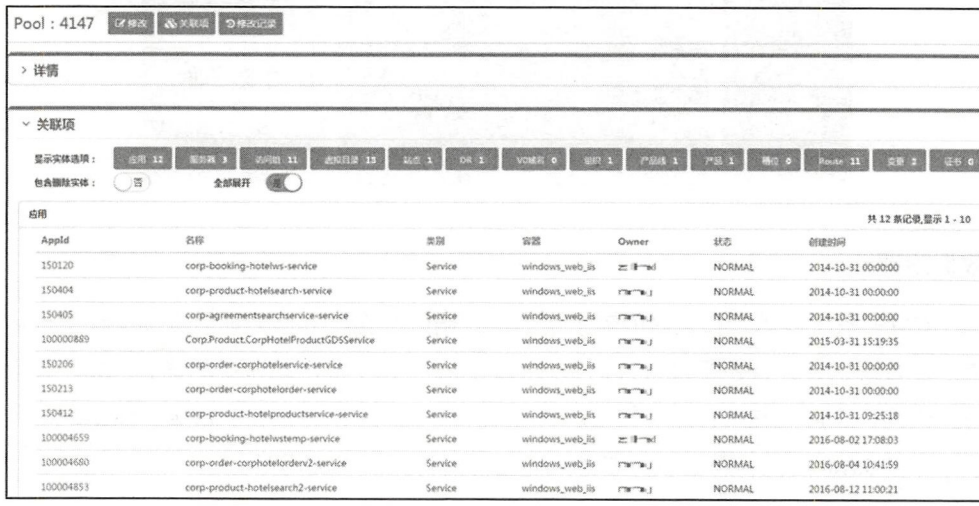
所以为了保证数据的准确性，CMS 数据需要持续治理。我们把这部分的治理工作通过一个相对独立的规则引擎来实现。该规则引擎主要完成的工作包括：

- 允许快速添加新规则，可以使用轻量的脚本语言快速定义各种规则进行数据检查。
- 针对复杂规则设计了场景和规则两层结构，不同场景可根据需求来配置不同规则。
- 数据入库时做检查，并进行定期巡检，最大限度查找和消灭错误配置。

关系管理和变更追溯

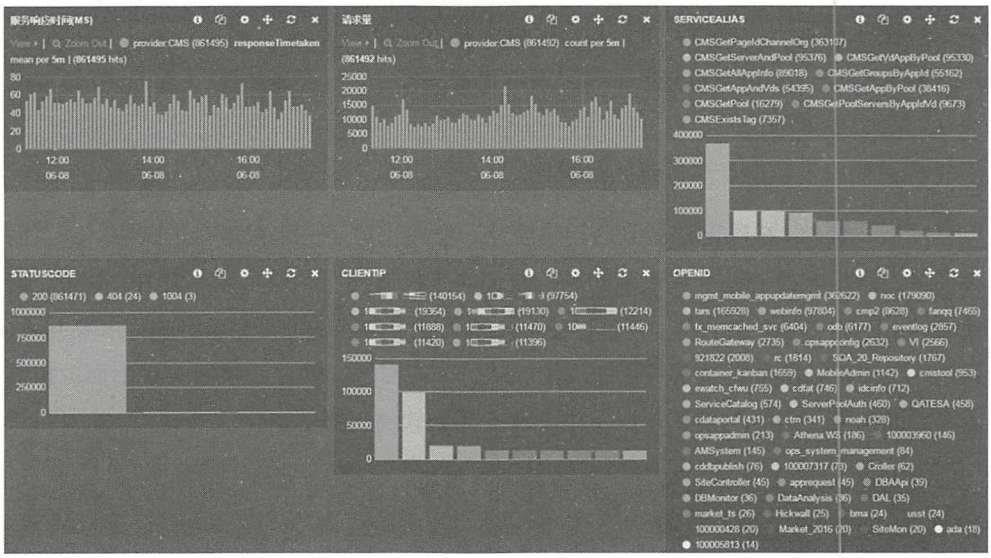
对配置数据关联关系的管理和使用是 CMS 用户最为看重的功能之一。被 CMS 管理的组织、产品、应用、集群、服务器、域名、发布节点等配置对象间都有着千丝万缕的关系，用户可能从任何一个配置对象开始查找与另一个配置对象的关系。比如从应用查找服务器，从服务器查找组织，从域名查找应用等。为了提供最便利和强大的查找功能，我们专门设计了一套查询框架，根据定义好的对象关系快速生成配置对象之间的查询。现在用户可以通过 CMS 界面和 API 非常方便地查找到配置数据间的关联关系。

与此相关的还有变更历史的查找，用户除了需要查找一个配置对象自身的变更历史，还经常需要查找一个与配置对象相关的对象变更历史。比如要查找一个应用下面所有服务器的扩容或缩容历史，查找一个集群中应用上下线的历史等。于是我们采用了一种将变更消息沿对象关系链广播出去的方案，把变更和相关配置对象连接起来。



完善的监控和应对访问压力

CMS 因汇聚了生产环境核心的配置数据而被大量工具所依赖，因此其必须能够承受大量而密集的查询需求（工作时间内每分钟上万次请求是常态）。下图是我们接口网关日志分析出的各种工具对 CMS 接口的调用情况。



弹性路由（SLB）

我们部署架构采用的是单机多应用，每台服务器上部署了很多个应用。这些应用不一定存在紧密的内在关系，并且很可能属于不同团队，这种架构存在着明显的问题。

其实我们面临的这些问题并不是突然暴发的，而是经过十多年的演进和慢慢累积，最终大家不得不正视这些问题。从本质上讲，这些问题的根源是应用间的耦合，最好的解决方案就是单机单应用。因为单机单应用实现了应用间的天然物理隔离（部署在不同的服务器上），从而极大地降低了运维的复杂度，部署、排障、沟通、配置和个性化等都不用再担心会对其他应用有影响。

单机单应用是业界普遍推荐和采用的一种部署架构，但对我们而言，这却是个系统性的大工程，需要从底层基础设施到配套系统工具，从流程规范到开发人员的思维转变等方面投入大量的人力和时间。所以我们首先就要考虑如何在单机多应用的情况下，实现应用解耦，也就是做到应用粒度的运维。

相比应用粒度的运维目标，我们当时的实际情况是服务器的运维粒度，并且绝大多数的运维操作还是通过硬件 LB 来完成的。虽然硬件 LB 的好处显而易见，例如，高吞吐量、高性能和优秀的稳定性等，但其缺点也同样明显。

- 水平扩展成本高昂。

- 基于规则无法建模，规则过多时就会陷入运维泥潭。
- 无法进行高频次的变更，因为集中式管理模式中，配置数据一多，API 性能就会急剧下降。
- 只能由少数的专职运维人员操作。

所以，硬件 LB 除无法做到应用粒度外，低效也成了一个很重大的缺陷。为了解决在路由运维方面的粒度和效率问题，我们决定打造自己的软负载（SLB）系统，替代硬件 LB 的 7 层路由职责。经过讨论，SLB 确定了自己的职能目标，即可以高并发、实时、灵活、细粒度调整 7 层路由规则。从另一方面想，SLB 还需要实现由面向机器运维到面向应用运维的转变，以及由硬件支撑到软件支撑的进化。

在我们 SLB 的开发过程中，最重要的几点是：

- 面向应用建模。
- 多次更新一次生效。
- 多并发操作的挑战。
- 多角色运维冲突的问题。
- 监控和告警。

面向应用建模

我们经过评估，最终选择了 Nginx 来构建软负载系统。开发前我们参考了业界内其他公司的实现方式，基本包含以下几个特点：

- 开发了一个 Nginx 配置文件的批量管理工具。
- 需要专业的运维人员来操作。
- 日常操作频率较低。
- 和现有系统接合较松散。

结合我们的现状，我们在建模时还需要考虑：

- 和现有系统无缝接合，融入现有系统的生态体系。
- 支持高频率的并发操作。

但如何与现有建模体系融合起来呢？在开发人员眼中，最重要的核心常见模型就是一个个的应用。所以 SLB 要做的是如何和应用模型融合起来，换句话说，所有对

SLB 的操作都要被抽象为对一个应用的操作。Nginx 是基于文本配置文件，其内建了一个自己的模型，一次运维操作可以导致多个 Nginx 模型的变更。所以我们需要创建一个模型，这个模型可以和应用模型一一对应，又能被翻译成 Nginx 的内建模型，而这就是 Group。

- 一个 Group 是一个应用在 SLB 的投影。
- SLB 上所有的操作都抽象成对 Group 的操作。
- 不同 Group 的操作互不影响。

这样只要解决一个 Group 的问题，就相当于解决了 1000 个，甚至更多个 Group 的问题。

多次更新一次生效

建模成功地隐藏了 Nginx 的内存模型，并将操作转换成了对 Group 的操作。虽然隔离了不同 Group 间的操作，但在 SLB 上对单一 Group 的操作仍然是一个有风险的行为（对某一具体应用而言）。为了降低这种风险性，可以引入 3 种机制，包括多版本系统、日志追踪和多次更新一次生效。

Group 的每次变更都会产生一个新的版本。Group 的所有变更都会留下日志。对 Group 的变更操作并不会直接对生产生效，可以在多次变更后，有一次明确的激活操作后，从而在生产环境正式生效。

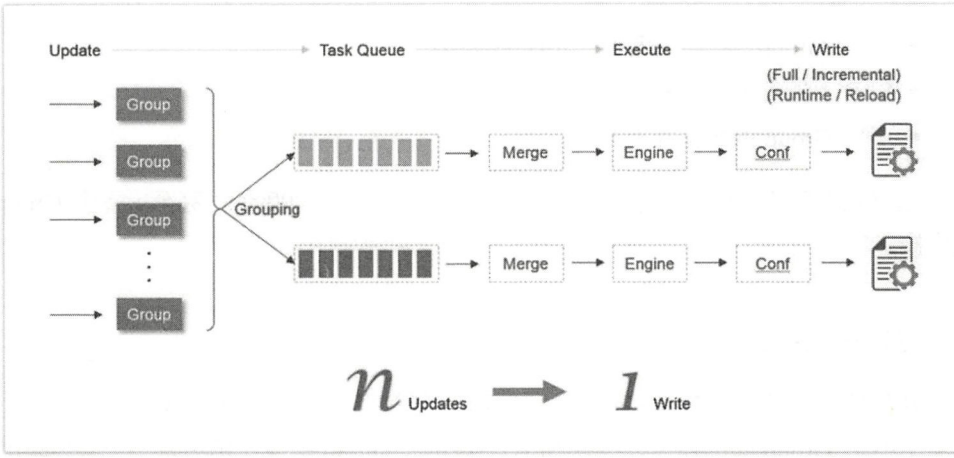
多并发操作

引入 Group 后实现了应用的独立运维，但如果有上千个 Group 同时进行扩容操作，那么如何做到每个 Group 的操作都在 5 秒内完成呢？因为 Nginx 是基于一个文本配置文件的，那么这样的要求就会转换为对配置文件的上千次操作，然后再对 SLB 重新加载上千次配置文件。假设一次操作花费 1 秒，那么最后一个操作可能要等 1000 秒，这种实现方式显然对于那些排在后面的 Group 更新者是无法接受的，而且 SLB 在这种高频度更新下，自身也无法工作。所以简单地把一次 Group 更新转换成一次 Nginx 的配置更新肯定是行不通的。

我们的真实情况是 Nginx 变更日操作达到 8 万次，整个软负载 API 日请求数达到 300 万次。为了实现 Group 更新的互不影响，并确保所有 Group 更新保持在一个稳定

的返回时间内，SLB 确定了核心业务流程。

- 将一段时间内所有的 Group 更新操作（比如 2 秒内）缓存在一个任务队列中。
- 对任务队列中的所有操作进行合并，最终只对 Nginx 的配置文件做一次更新。



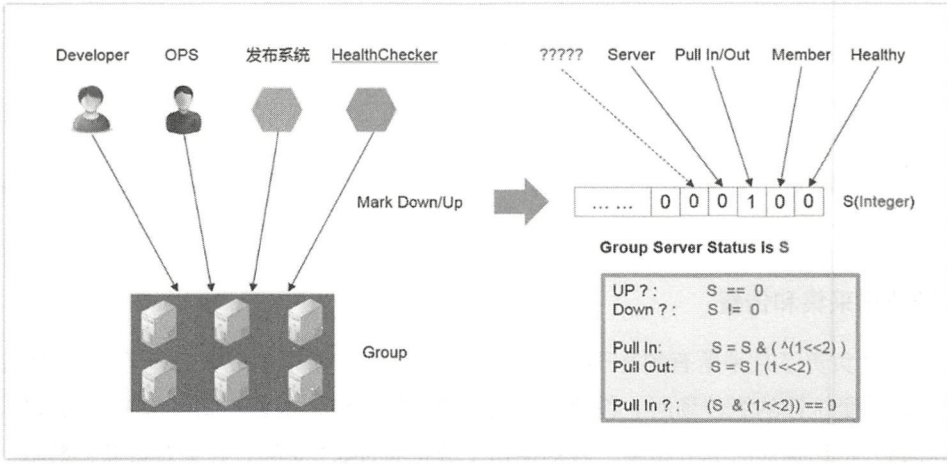
这个流程的核心逻辑就是多次操作一次更新，最大程度减少对 Nginx 配置文件的操作。但外部看来，Group 更新操作是独立且保持在稳定返回时间内的。

多角色运维的冲突

一个 Group 可能会有多种角色进行更新，比如应用 Owner、专业运维人员和发布系统人员等。这就引出了一个新的问题，当一个角色对一个 Group 的服务器进行拉出操作后，另一个角色可不可以对这些服务器做拉入操作？比如，发布系统人员在发布完成后，准备做拉入，却发现运维人员对这台服务器进行了拉出操作。这时发布系统应该如何决策呢？这不仅造成决策的困扰，也会使不同的角色产生联系，甚至相互耦合在一起。

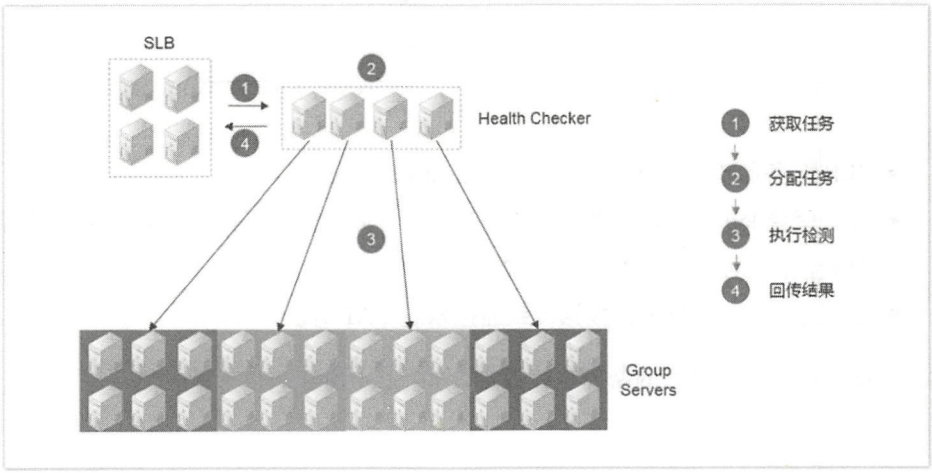
为了解决这个问题，我们决定采用多状态的机制。

- 为每一种角色分配一个服务器状态。
- 一个角色对这个状态进行了失效操作，最终也只能由这个角色进行恢复操作。
- SLB 在所有角色都认为这台服务器有效时，才会认为这台服务器可工作。

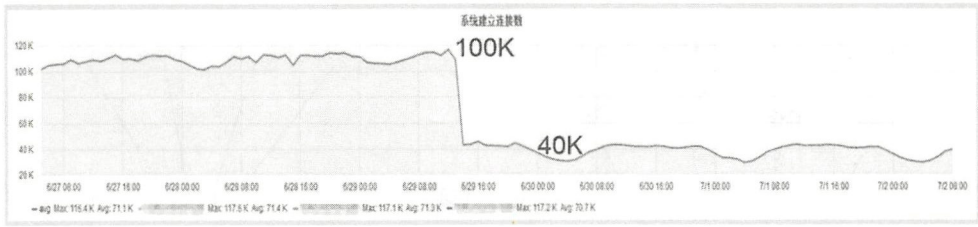


健康检测带来的瓶颈

SLB 另一个核心功能是健康检测，即需要以一定频率对应用服务器进行心跳检测，连续失败多次后对服务器进行拉出操作，成功后再进行拉入恢复。大多数公司采用了节点独立检测，造成了带宽浪费和服务器压力，而我们采用了节点共享检测，具体机制是一个独立的应用负责检测，然后把检测结果在 SLB 节点间传播共享。

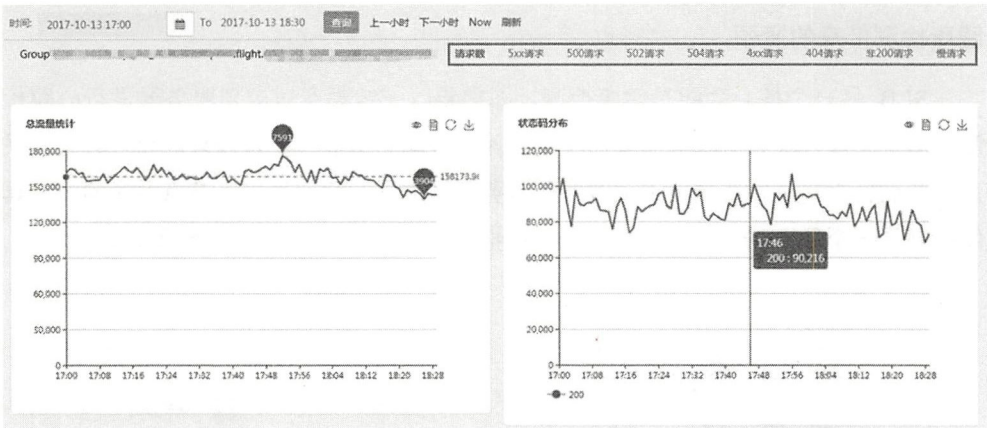


我们独立健康检测的运行效果良好，目前 SLB 系统已经负责了我们超过 5 万个结点的健康检测任务。而下图是由节点独立检测变为节点共享检测时的 SLB 单一服务器网络连接释放状况。



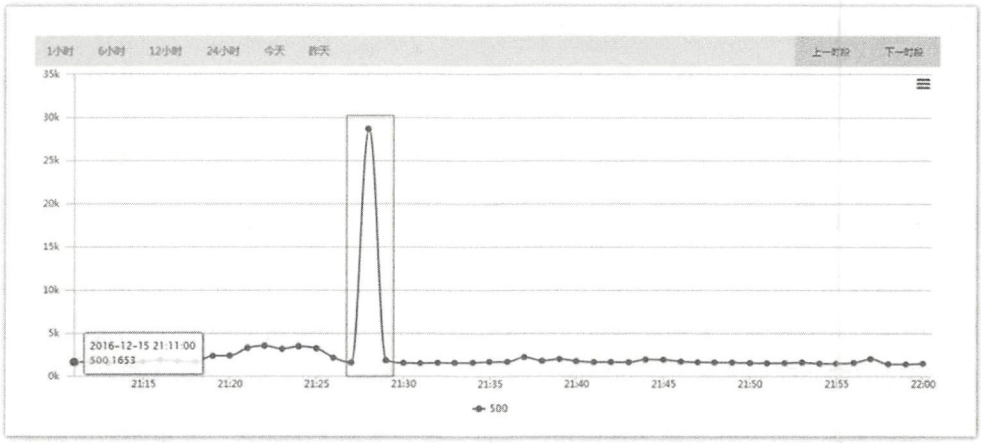
监控数据采集和告警

SLB 负责了几乎所有的基于域名的 http 调度请求，所以也成了进行请求流量统计和请求质量统计的绝佳场所。包括在有问题时进行报警，根据不同维度统计请求量，响应码分布和响应时间分布等。我们使用了分析 access log 的方式来获得监控数据。



- SLB 服务器流式读取本机实时产生的 access log。
- 分析聚合 log 数据，产生不同的统计数据。最终使用了语法树分析实现了高效分析，1 秒可以分析 14 万条日志。
- 定期（1 分钟）将统计数据传到监控系统 CAT 等。

以此可以产生多维度的监控统计数据，如下图所示。



基于上述数据，可以查看全部或单个应用性能表现，进行相应的优化。在慢请求和非 200 请求的数量异常时，执行报警操作，确保及时恢复和挽回损失。

想发就发（TARS）

解决了配置和路由问题后，发布系统前置障碍已基本扫除，而从 OPS 角度来看，发布系统还有几个重要目标：

- 灰度发布
- 简单易用
- 发布迅捷

发布方法

通常发布有三种常规方法：蓝绿发布，滚动发布，金丝雀发布。对这三种发布方法做比较，可以发现：

1．蓝绿发布。需要额外的服务器集群支持，且数量可观，同时由于我们单机多应用的部署现状，就会造成发布一个应用需要替换整台服务器的情况，实现难度巨大且成本不经济。

2．滚动发布。虽然可以节省资源，但对应用的兼容性有较高要求，因为发布过程中同时会有两个版本对外提供服务。但这类问题相对容易解决，实际中往往会通过

功能开关、dark launch 等方式来解决。

3．金丝雀发布。比较符合我们对灰度发布的预期，但可能需要精细的流控和数据的支持，同样有版本兼容的需求。

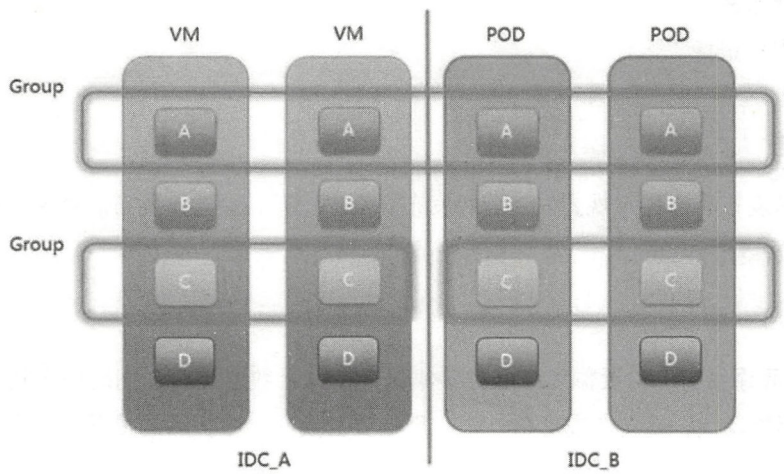
发布相关说明

蓝绿发布：优先将新版本发布到待发布的机器上，并进行验证，此时新版本服务器并不接入外部流量。发布和验证过程中老版本所在的服务器仍照常服务，验证通过后，经过流控处理把流量引导到新服务器，待全部流量切换完成，老版本服务器下线。

滚动发布：从老版本服务器中挑选一批，停止老版本的服务，并更新为新版本，进行验证，通过后再分批增量更新剩余服务器。

金丝雀发布：往往从集群中挑选特定服务器或一小批符合要求的特征用户，对其进行版本更新及验证，随后逐步更新剩余服务器。

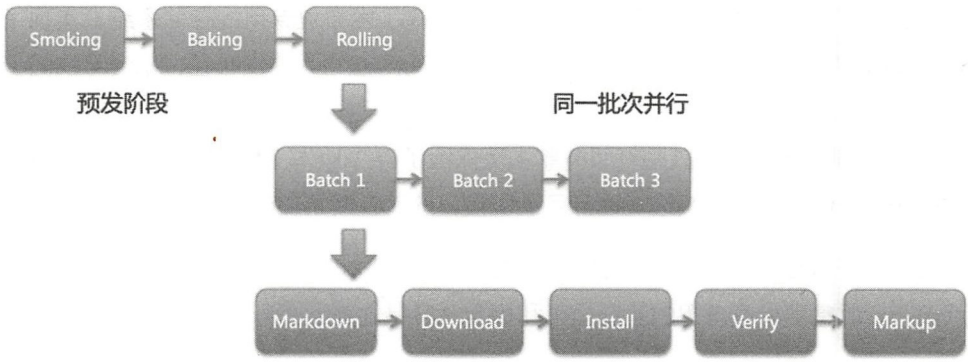
结合我们的实际情况，最终挑选的方式是滚动发布和金丝雀发布的结合体，首先允许对一个较大的应用集群，特别是跨 IDC 的应用集群按自定义的规则进行切分，形成较固定的发布单元。每个应用的每个发布单元称为“Group”，这个 Group 与之前提到的 SLB 的 Group 是一一对应的。



每个发布单元，即 Group 在发布过程时，还可以再分批进行，完成滚动发布。而每个 Ggroup 中包含一台或多台堡垒机，必须先完成堡垒机的发布和验证，才能继续

其他机器的发布，从而实现金丝雀发布。除堡垒机的发布外，其他机器可按照用户能接受的最大同时拉出比例来分批，分批间允许设置具体的验证等待时间。

每台机器在发布过程中都要经历拉出、下载、安装、点火和拉入这 5 个步骤，发布流程如下图所示。



基于以上设计，我们新一代发布系统开发完成，命名为 Tars。Tars 已做了开源，开源版本地址为 https://github.com/ctripcorp/tars。

简单易用

发布配置必须简单易用，绝大部分的应用发布都是固定模式，不需要个性化配置，所以 Tars 只提供了几个核心配置项，包括允许同时拉出的最大比例、批次间的等待时间、启动超时时间和是否忽略点火。

除此以外，用户最关心的是发布过程中可操作按钮的易用性，Tars 在这方面做了充分考虑。通过状态机的控制，保证用户在操作界面上同时最多只看到两个操作按钮，绝大部分情况下用户只需在“继续”或“终止”这样的 0 或 1 的选择中做出决策。一步一步迭代出无线持续交付平台，展示图形化界面，Tars 也确保用户可以更直观地观察到发布的进展，以及出现的问题。

有了简单操作，危机时刻就会得到放大体现，比如，因生产故障做回滚时，能快速中断当前发布，并从界面中轻松地选到所需回滚的版本，然后一键无配置地触发完成回滚。

发布迅捷

天下武功无坚不摧，唯快不破，发布也一样。发布速度快了，迭代速度研发效率也就提升了；回滚速度快了，生产故障造成的影响也就减轻了；扩容速度快了，弹性计算就能实施了，这样运维效率被大幅度提升。从上面对发布过程的描述中，不难发现通常影响发布速度的步骤是下载和验证。

1．为了提高下载速度，我们在各个机房搭建了发布包专用的存储系统，实现了类似 CDN 的功能，编译和打包系统在任何一个写入点写入的发布包，都会尽快同步到各个 IDC 及各个独立存储中，这样真正发布时，服务器只需从本 IDC 或本网段做下载。而回滚方面，Tars 则是在服务器本地保留了 *N* 个版本（*N* 根据服务器磁盘容量计算获得），做回滚时可快速地进行目录切换，进而省略了代码下载过程。

2．对于验证，我们在框架层面统一提供了验证入口和常规验证方法（我们称为“点火”），收集了所有应用的验证规范和标准，容错性得到提升。

3．Tars 在系统设计方面充分考虑了速度需求。每个发布单元采用 quick and dirty 的方式，不管成功或失败，优先尝试把版本发布完成，后续再解决个别发布失败的问题。根据同时拉出服务的最高比率（由用户设置）进行失败率控制，一旦达到比率，立即中断当前发布，从而对 quick and dirty 方式做保护（我们称为“刹车”）。发布单元中只要有任何一台服务器发布失败，都会被认为是发布局部失败，允许用户重试发布。发布过程中如发现服务器当前运行版本与发布目标版本一致，且验证通过，则直接跳过。批次间可设置观察等待时长，从第 3 个批次起，允许设置 0 或较少的等待时长，以提高后几个批次的速度（我们称为“尾单加速”）。

结果和未来

通过 CMS+SLB+TARS 几个系统的联动，并经历了长达一年半的项目推广阶段，终于实现了“1+1+1>3”的效果。新发布系统对于研发效率和研发人员体验的提升都非常显著。

这可以通过一些数字来证明，与两年前相比，每周的发布迭代次数增长了 4 倍，单次发布的平均时长从 13 分钟降低到了 3 分钟。同时因为发布/回退效率的提升，当需要对线上代码做紧急修复时，或者将其回退到已发布的代码版本时，都会更快捷地完成，所以使得发布类故障的处理效率也得到了提升。对 2015 年至 2017 年的发布相

关故障统计后，发现该占比下降了一半以上。

因为 CMS+SLB+TARS 基于良好的配置数据模型设计，以及其应用级的运维支持能力，为后续的技术架构改造带来了便捷和优势。这主要体现在：高效的容量管理，实现了对应用容量的自动化监测，当发现容量不足时，无需研发人员介入，全自动地进行应用服务器扩容、发布、上线和投产等。在应用容灾方面，基于准确的配置数据，可以很容易地将单数据中心的业务应用“克隆”到另外的数据中心来进行部署。

在应用技术栈的迁移（例如.net 应用改造为 Java 应用）中，用户也能自助地创建新的 Java 应用，并通过 SLB 灵活实现灰度流量切换，进而自助、高效、稳定、安全地完成整个应用迁移。

5.1.3　一步一步迭代出无线持续交付平台

目前我们大部分订单来自移动端，APP 几乎承载了整个集团的所有业务形态。在内部研发中，我们的 APP 已经发展成为拥有 90+Native Bundle，100+Hybrid Bundle，30+RN Bundle，几百名研发人员，每个版本（1 个月）交付 4000 多个 APP 包，Hybrid、RN、Hotfix 和 Bundle 发布次数超过 500 次。很难想象，如果没有一个有效的无线持续交付平台，很难在 3 天内完成大版本的集成发布。

市场上开源的无线持续集成工具 fastlane、test flight、Jenkins 都存在着各种定制化需求的问题，因此我们从零开始，逐步打造适合我们研发流程的无线交付平台，系统化地解决研发支撑痛点。下面将从集成、测试、发布、运营 4 个子平台来展开分享我们是如何一步步打造无线持续交付平台的。

集成平台

从最初到现在，我们无线持续交付模型经历了从 1.0 到 2.0 的迭代，在 1.0 之前 APP 集成和发布还主要依靠人工操作 Jenkins，需要由特定的打包人员负责打包，再将包通过 IM/邮件等方式传递给其他测试人员，测试结果需要专人手工回收来把控 APP 质量。此时最大的问题就是 APP 管理混乱，人工介入过多，每次发布都需要花费很长的时间。

1.0 阶段

1.0 阶段我们引入 MCD（Mobile Continuous Delivery）平台思路将打包人员的工作交给我们的平台来完成，提高了发布工作效率。这时不需要专职的打包人员负责出包，平台会定时自动打包，测试人员可以到平台上自由取包（通过下载链接或者扫描二维码）进行测试，与此同时测试人员也可以在平台上进行单独的打包和测试。测试结果也会统一反馈到平台上，协调人员在平台根据各家的反馈结果决定需要重新出包还是继续下一步发布操作。

在这个阶段，APP 的打包还完全依赖于源代码，由平台生成打包参数（主要包含 APP 运行环境、与 iOS 签名相关的参数以及代码仓库相关的参数）提交给后台的打包系统，打包系统根据仓库地址和 commit ID 从代码仓库中拉取全量代码，打包系统再调用代码中预先准备好的 Build 脚本构建 APP 产物，构建完成后将构建结果保存到临时的文件服务中，最后由平台的回收进程将构建结果回收并处理之后放在云存储上供用户下载使用。

2.0 阶段

1.0 阶段虽然已经基本实现了集成打包的自动化，但是还存在以下几点不足：

1．源码打包方式效率低下，采用源码打包每次都要从代码仓库中下载全量代码，再通过源代码生成出 APP 产物，这样做使得每次 Build 时间都很长，而打包次数增加导致某些打包任务积压，系统不能及时出包。

2．编译容易失败，任何一个 checkin 导致的编译失败，就会导致系统不能正常出包。

3．系统之间采用轮询模式，Job 任务扩展性差。

MCD 系统发起 Build 请求之后会有另一个定时的 Job 任务去轮询 Build 服务器查看 Build 结果。在初期阶段还能满足业务需求，但是后来由于打包数量的增加以及打包频率的提高，系统的处理效率变得越来越低。一方面打包资源不够（Android 打包使用 Linux 虚拟机，iOS 打包使用 Mac），另一方面轮询 Job 的处理效率达到了瓶颈。打包机器采用 Jenkins 方式进行管理，因此很容易进行横向扩展，但是 Job 却很难扩容。

针对以上问题我们对平台进行了升级改造：

1．引入消息机制，提高系统吞吐量。

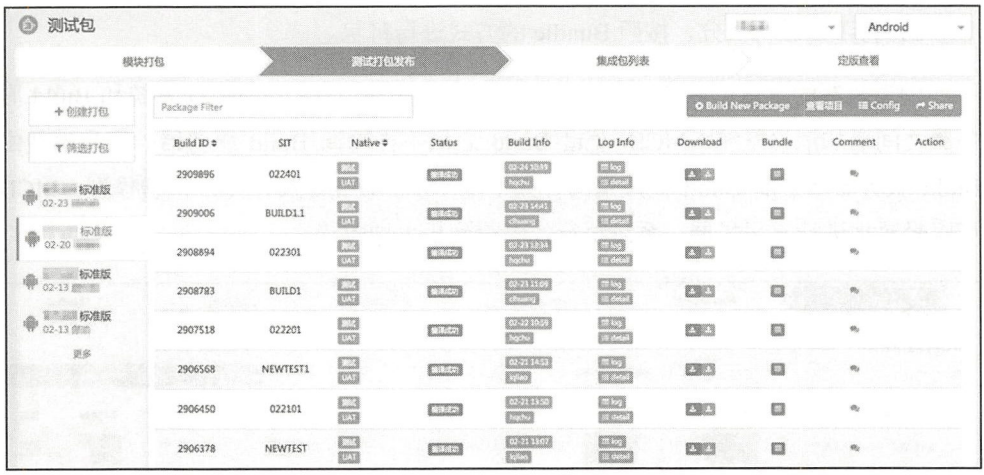
2．将工程进行拆分，按照 Bundle 的方式进行打包。

消息机制的改造：首先基本打包架构和流程保持不变，在 MCD 系统和 Build 服务器之间增加消息系统，MCD 发起 Build 之后不再轮询 Build 服务器，而是消费由 Build 服务器产生的 Build 完成消息，如下图所示。使用这样的生产消费模型，MCD 可以轻易地进行水平扩展，系统执行效率得到极大地改善。

工程拆分：APP 工程拆分成多个不同的 Bundle 模块，Bundle 之间存在依赖关系。在这个情况下，APP 的编译和打包也可以按照 Bundle 的方式进行组织。在开发阶段，开发人员提交完代码之后就能直接将自己负责的 Bundle 编译成可执行文件。测试人员可以自由选择 Bundle 进行打包，此时打包工作只需要将多个 Bundle 文件组装在一起就行，极大地加快了打包速度。通过测试的 Bundle 会被发布到指定地点，在 APP 集成阶段只需把所有发布的 Bundle 组装成最终 APP 供大家测试即可。

在开发自测阶段开发人员提交完代码之后，在 MCD 平台上 Build 出所开发的 Bundle，并标记为 L 版本（表示 Latest）。相关测试人员（或者是开发人员自己）可以打测试包，测试包会使用所有 Bunde 的 L 版本进行构建。构建完成之后获取测试包进行功能测试，测试通过后就可以将该 Bundle 进行发布操作，即标记为 RC 版本（表示 Release Candidate）。

在集成测试阶段，MCD 平台会自动选取已发布（RC 版本）的 Bundle 打出集成包。测试完成后，测试人员可以将测试结果反馈到 MCD 平台中，待测试全部通过之后就可以安排 APP 进入发布定版阶段。再进行一些后续市场包操作，就可提交给 APP 应用市场供用户下载。



由于 APP 的开发是个连续的过程，测试包和集成包都可以很方便地配置是需要 hourly build 还是 daily build，并且可以和测试平台的相关功能联动，从而实现持续集成。

我们也会将不同版本（不同功能）的 APP 按照项目进行划分，主版 APP 为标准项目，非主版（渠道/Hotfix/Bundle）APP 为非标准项目，同一项目内的 Bundle 可以自由组合，项目之间互相独立，不受影响。一个项目在创建的时候可以选择从另一个项目继承其各模块的最新版和发布版。项目在开发过程中也可以从其父项目中实时同步对应模块的最新版和发布版，这样就能基本满足各种开发和测试的需求。

测试平台

随着无线持续交付的发展，原有的测试模式以及流程渐渐显现出不足，主要体现在以下三点：

1．快速验证与手工验证的冲突。

2．自动化回归测试与手动测试效率的矛盾。

3．设备兼容测试与设备不足的矛盾。

快速验证与白屏检测

在集成测试阶段，每一次出包首先会由基础测试人员负责验证此次出包的质量，主要是验证各个 BU 入口页面（Hybrid 或者直连页面）是否正常。如果大量 BU 入口页面

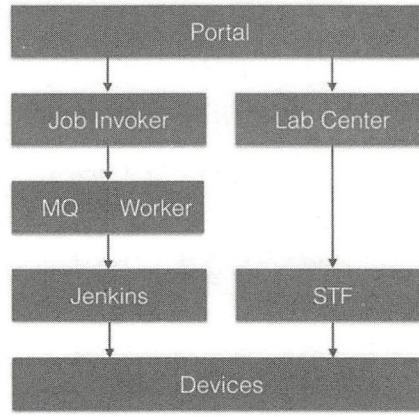
异常，则会要求重新出包。这个过程看似很简单，但是（沟通成本+验证成本）X 出包次数所耗费的时间也是不容忽视的，而且这种冒烟测试对于测试人员也很枯燥无趣。

针对以上情况，最初引入了白屏测试等基础性测试功能，并与集成平台打通（如下图所示，可以直接从集成包列表选择相关测试功能）。白屏测试主要是检测 APP 首页各个入口页面是否显示正常，对于每一个页面会进行图像以及页面长度比对来验证页面是否正常，会在多台设备进行白屏测试，只要其中一台设备通过，则认为这个入口是正常的。

自动化回归测试

通过自动化回归测试，可以提高回归测试效率，有效缓解测试人员的压力。我们逐步开发了 MCD 测试平台用于自动化测试项目的管理、执行、报告和展示，支持 Android/iOS APP，H5/Hybrid testing。

测试平台架构简化图如下图所示，各个模块主要功能如下。



1．Portal 负责与用户交互，查看报表，保持后台系统对用户透明。

2．Job Invoker 是调度通道，用户通过 Portal 提交的测试任务通过此通道来调度，包括创建项目、设备占用、测试项目执行等。

3．Lab Center 负责所有与设备相关的业务，Lab Center 与二次开发的开源软件 STF 交互，所有设备挂载在 STF 上。

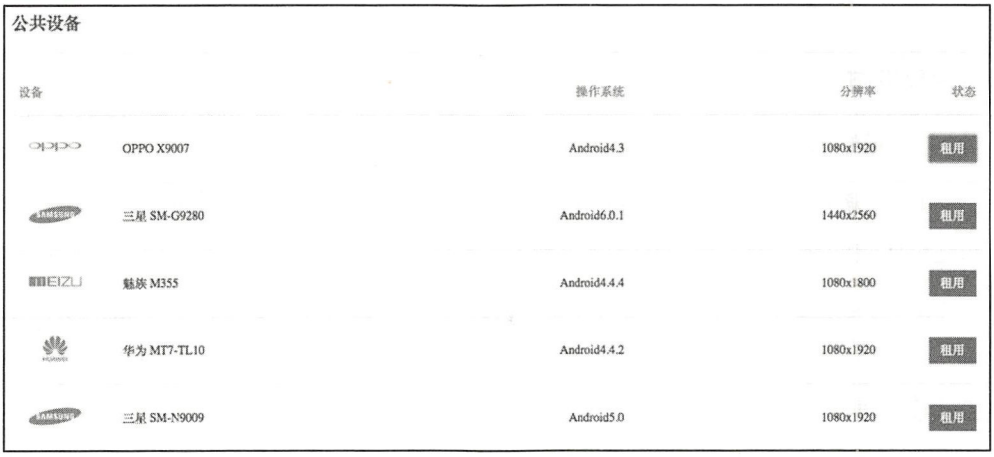
4．当具体的测试项目在 Jenkins 上执行时，会将预先占用的设备远程连接到 Slave 上等待使用。

MCD 测试平台提供了自动化测试（白屏检测、录制回放、回归测试）、系统测试（兼容性、性能、稳定性）、手动测试（远程租用、远程调试）三大类功能，基本达到了我们对于提高测试效率的需求。

core_case_monitor_p项目Run列表

Run ID	任务状态	策略	执行人	Tags	任务APK	执行设备数	Task数量	创建时间	结束时间	用例数	通过数	失败数	通过率
106376	finish	specific	[redacted]店研发部)	android specific		1	1	10-24 07:33:01	10-24 11:07:17	90	78	12	86.67%
106366	finish	specific	[redacted]店研发部)	android specific		1	1	10-23 15:28:00	10-23 20:38:05	90	78	12	86.67%
106361	finish	specific	[redacted]店研发部)	android specific		1	1	10-23 07:33:01	10-23 14:41:36	90	0	90	0.00%

另外日常测试中还经常遇到这些情况：需要快速验证 APP 的一个功能，有些问题只能在特定机型上重现，测试目前却没有这款手机。因此我们基于 STF 二次开发出设备共享平台，将空闲设备收集起来在平台上共享，用户只需在平台上搜索需要的设备就可以立刻通过 STF 进行使用了。



我们也建立了代码质量分析功能的自动化，集成了 Sonar、Facebook Infer 和 Uint Test 功能，方便每个版本对代码质量进行跟踪。

发布平台

无线发布系统一直以来都是无线应用能快速迭代的关键一环，一个好的发布系统能帮助开发人员快速地修复线上问题，并能快速地将新功能投送至用户，帮助业务抢占市场，而发布系统的设计也随着 CD（Continuous Delivery）概念的深入人心而日新月异。

无线发布的特点是静态资源包（不管是 Hybrid、RN，还是 Hotfix、Bundle），发布技术中涉及以下主要问题：

1．如何最大限度地把静态资源的下载流量降到最低，以减少用户对应用更新的感知度，提高应用升级的成功率和用户体验。

2．灰度发布，保证发布的质量。

资源包发布

针对静态资源包发布，经历了如下几个历程：

1．全量包发布：我们最初使用的都是全量发布的方式，这种方式实现简单，而问题也比较明显，导致用户升级时花费流量巨大。

2．文件字符串差分：这种方法避免了整包差分的问题，能够比较方便地实现快速迭代要求下的小量更新需求，并且文本差分工具简单，实现容易，出错较少，但是差分效果差。

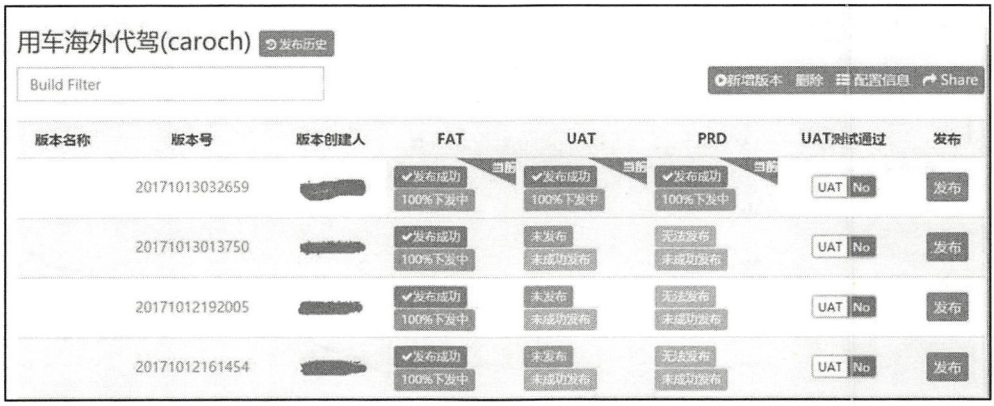
3．文件二进制差分：对每个文件进行二进制级别的差分，能够具备小量快跑的特点，同时进一步优化差分文件 size。这也是我们目前使用的方式，它仍然使用了 bsdiff 作为差分工具，当文件没有更新的时候，不输出差分文件，而更新时也只产生 size 非

常小的 diff 文件。

4．按需差分：在按需差分前，我们采取过的方案是预差分方案，就是在发布时，把所有版本间的差分准备好，有多少个历史版本就将产生多少个差分包。这种做法将随着发布次数的增多，发布的差分包数量也急剧增长，对存储的要求将越来越高，也造成了浪费。

在此前提下：引入了按需差分，就是终端用户通过将自己的版本号与线上最新的版本号做比较，发现落后时，触发差分过程，从而获取增量更新。

这里会有一个问题：终端用户第一次访问时没有差分包。我们的做法是：第一次只是返回全量包，用户不用等待，但是后续用户会拿到差分包。另外一个问题是：避免重复打差分包。在计算时发现无差分包，会触发差分包过程，同时会把这个信息缓存（过期时间为半个小时），当其他客户端有同样的版本请求时，会先判断缓存，没有的话进行差分操作，有的话直接下发全量包。



灰度发布

网站发布通常有灰度发布的过程，无线发布与网站发布的流程是类似的，但是做法有差别。二者的区别是：网站是 centralize 的，可以统一控制，而无线发布是 decentralized 的。无线发布包，像 Hybrid、RN、Hotfix 等静态资源，是需要客户端一个个拉取的。如何确保整个发布的流畅，确保发布质量呢？我们的解决方案如下。

1．Offline 灰度发布流程：灰度第一步是冒烟，网站的冒烟是导入一部分流量到某台服务器上，与网站的冒烟不同的是，我们通过配置一个白名单，只有客户端在白

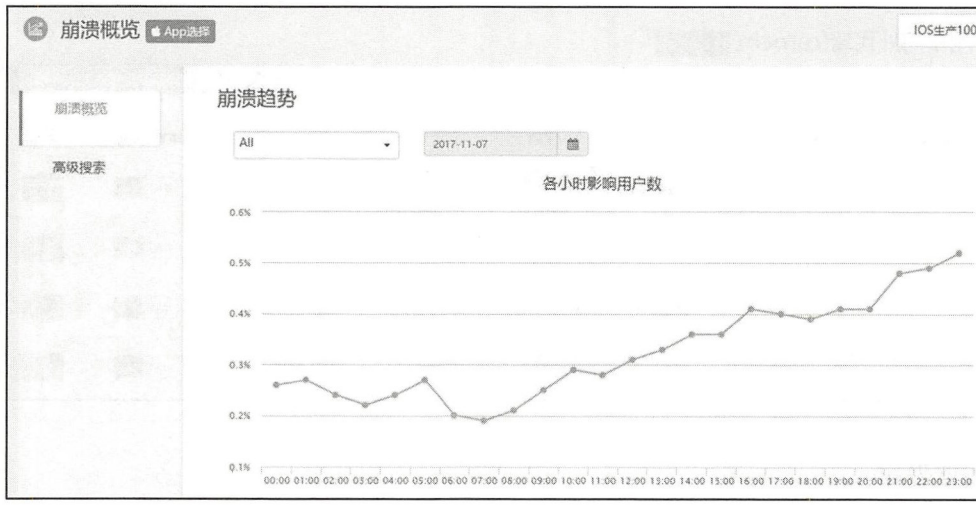
名单中才会获取到最新发布。当测试没有通过，可以选择终止，然后再决定是否需要回滚，测试通过决定是否需要扩大比例。

2．监控发布数据：当客户端收到下载物后，会把当前的下载信息上报上来，供发布人员进行监控查看。

3．灰度发布设计中需要注意：避免一直是小白鼠，为了避免一个用户一直当小白鼠，每次灰度需要利用灰度 ID+客户端信息进行取模运算；灰度回滚，灰度回滚时只需要回滚已经更新这次灰度的客户端，对于没有更新的，则无需回滚。

运营平台

运营是研发流程的最后一环，我们的 MCD 平台思路是对发布结果、运营情况以及配置等功能有一个集中的管理，包含运营看板、性能看板、发布看板、无线配置、崩溃管理、APP Size 统计等模块。



其中无线配置经历了多次迭代，对于各种配置项，从版本、平台、渠道、AB 多个角度进行了定义，做到灵活动态配置，粒度大小自由配置。下发时根据 APP 的版本、平台、渠道等信息挑选符合条件的配置进行发布。需要修改的时候也是先修改测试环境配置，测试通过后再同步到生产。配置分为专用版和通用版两种，专用版只会下发给特定的渠道/版本，通用版以版本号为基准，在不发布新版本配置的情况下系统默认给 APP 端下发最近的通用版配置。当同时存在符合条件的专用版和通用版的时候，只

下发专用版配置。

崩溃管理也是无线运营的重要一环，目前行业内的 QQ Bugly 平台崩溃管理功能已经很强大，但是由于定制性需求我们还是开发了自有崩溃管理功能，方便快速发现线上问题，从而能够通过 Hotfix/Bundle 方式及时修复。

5.2　敏捷运维

以开发为中心的人通常认为，软件变化会带来回报，企业依靠其应对不断变化的需求。我们有太多的方法论和实践来指导软件开发的敏捷，卓越的团队也取得了很好的效果。实际上，在整个软件系统生命周期过程中，运维工作应该占了相当大的一部分时间，人们越来越意识到传统意义上的开发行为和运维行为存在脱节现象，从而导致冲突和低效，因此敏捷运维应运而生。

随着互联网技术的迅速发展，运维的事务也日益复杂，如何能更加高效地协调好人、工具与流程之间的关系，把运维人员从低效率、高强度、易犯错的人工操作彻底解放出来，让他们的能力与精力有更大程度的发挥，将是一个很大的挑战。

5.2.1　运维工作流平台的演进之路

我们的运维工作流平台经过三年时间的演进，从最开始引入商业产 BMC Remedy ARS 作为底层单一工作流引擎，慢慢演化到抽象工作流平台的建设并扩展支持更多开源的工作流引擎，同时把原来的平台进一步分层，建立了统一标准的接口，对业务进行了服务标准化、业务流化以及流程自动化的改造。

本节将从以下几个主要方面分别阐述流程平台的建设与演进。

- 面临的挑战
- 运维工作流平台的建设
- 收获总结
- 下一代流程平台的设计
- 未来展望

面临的挑战

在讲挑战之前，先简单看一下我们运维工作流平台的演进历史。流程平台的演进主要分为以下三个阶段。

第一阶段：摸索期

从 2013 年下半年到 2014 年上半年，我们引入了 BMC Remedy ARS 产品作为我们的 IT 服务管理工具，但是我们发现它基于工作流工作的思路可以借鉴到更复杂和核心的运维流程中，所以开始尝试使用它的底层引擎构建我们自己的流程。

第二阶段：成熟期

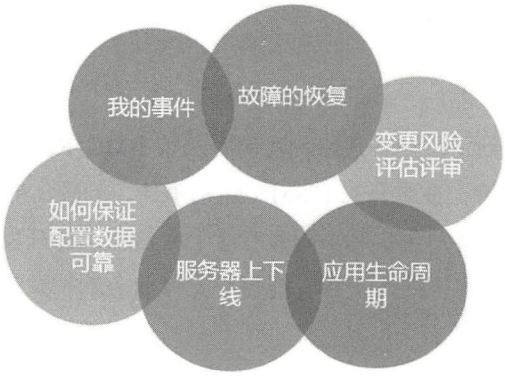
在 2014 年下半年到 2016 年上半年，在 Remedy 平台上相继开发了一系列的流程，包括服务器上线、下线，应用上线、下线、扩容、缩容、迁移等，还包括 ENP、API gateway 等一系列标准化的接口服务模块，开始渐渐形成流程平台。

第三阶段：革新升级期

在经过了一个成熟期之后，现有的流程也慢慢暴露出了一些新的问题，包括流程的可视化、价值数据的挖掘、底层流程引擎的单一。为了解决这些问题，我们从去年下半年开始，对原来的流程进行了重新设计，抽象出更加标准化的流程模型，以及标准规范的可视化和监控。

那么在没有流程平台之前，我们到底面临着什么样的挑战呢？在这里我给大家举一些运维过程中的常见例子。

比如日常工作中，当用户碰到问题后 IT 部门报障时，用户如何跟踪问题处理的进度，或者当网站发生故障时，如何能够快速恢复服务，后续的问题分析是否有流程支持，运维人员的日常变更操作是否有过风险评估以及审批授权，我们又是如何保障配置数据的可靠、准确性，服务器如何上下线，应用生命周期有没有统一的管控，等等。正是在这样一个背景下，我们开始着手设计一个综合的工具平台来统一管理运维过程中涉及的这些流程。

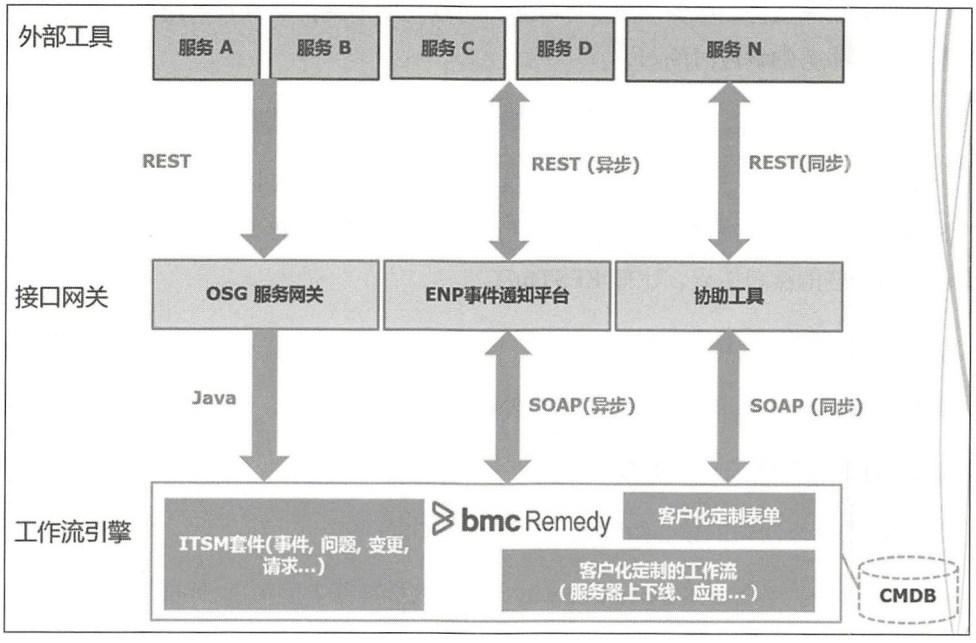


工作流平台的建设

关于自动化流程平台建设，自动化的意义相信大家应该都是有共识的，在上面的挑战中我们已经提到了这些运维过程中大家可能会碰到的问题，那么我们是如何去建设这个流程平台的呢？

1．系统架构

我们先来看一下最初设计的流程平台系统架构，如下图所示。



从下往上看主要有三层：

（1）底层工作流引擎

前面我们已经提到了，在一开始，我们引入了 BMC Remedy 产品主要作为内部 IT 服务管理流程的支持，在原有的 ITSM（包括事件、问题、变更以及请求单）基础上做了我们自己的一些客户化定制，同时在这个平台上设计开发了包括服务器上线、下线，应用相关的上线、下线、扩容、缩容等支撑业务的相关流程。

（2）中间接口网关

在中间接口网关，从左往后分别是 OSG 服务网关、ENP 事件通知平台以及协助工具。为什么会有这样的设计呢，主要有以下几点考虑。

第一，外部工具服务过多，需要统一标准化管理。需要对流程接口服务进行标准统一的管理，所有工具提供的服务需要在平台注册。

第二，服务访问权限的控制。可以通过平台对提供服务进行权限控制，可以查看提供的服务由哪些外部用户或者工具调用，同时申请了哪些外部工具的服务，可以对申请进行授权。

第三，工具提供的服务质量。可以通过可视化的前端页面查看当前提供的服务的质量，比如服务的响应时间。

第四，访问日志。当进行问题分析排障时，可以有被追踪的日志。

第五，产品局限。现有解决方案提供的接口只支持 JAVA 以及基于 Web Services 的 SOAP 协议，无法提供很好的扩展，所以需要在这层做些接口的包装，以支持更主流、灵活一些的接口协议，比如 RESTful。

（3）上层外部工具

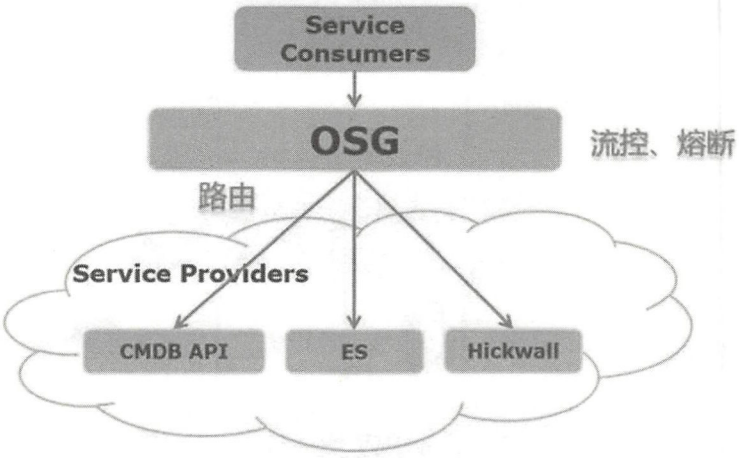
最上面就是需要来访问流程的所有外部工具服务。

2．标准服务网关——OSG

OSG 是接口网关的核心组件之一，什么是 OSG 呢？

OSG，全称为 Open Service Gateway（开放的标准服务网关）。外部工具作为服务提供者可以将他们的服务注册在 OSG 网关，其他工具若需要消费可以找到对应的服

务，以服务消费者形式申请某项服务的访问权限。

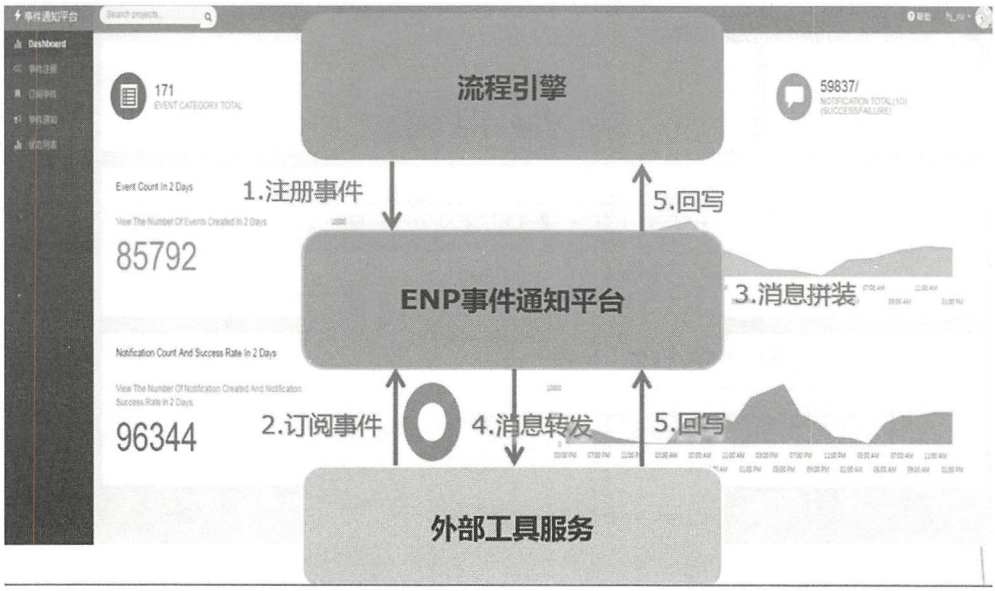


当服务与服务之间交互的时候，OSG 就充当着路由的角色，对访问的请求进行转发。OSG 另一个主要功能就是流控、熔断机制，可以为服务设置每分钟最大访问次数，当最大的访问次数超过设定阀值时，服务自动设置为 Blocked，直到服务提供者重新启用。

对于工具之间的服务相互访问日志，都会被集中采集并传到后台 ES 服务器，用户可以通过前端可视化的界面对服务的质量、工具的访问进行实时监控，同时可以根据日志进行问题排查。

3．流程与外部工具的桥梁——ENP

ENP，全称为 Event Notification Platform（事件通知平台），这里大家可能会有这样的疑问，这不就是消息队列吗？可以这么说，事件通知平台的设计参考了消息队列，但实现上又稍有些不同，之所以会有这个平台，是因为从 Remedy 产生的消息数据在处理上有些局限，所有数据的格式化处理都需要交给 ENP，由 ENP 根据设置规则进行封装处理后再转发给外部工具。



为什么需要这么一个平台呢？

刚开始在 Remedy 平台上开发第一个服务器上线流程的时候，整个上线流程中涉及了多个环节与外部不同工具交互。不同工具之间的接口实现非常复杂，所以在流程从半自动化到全自动化转化的过程中，开发人员花费了大量的时间和精力与工具进行联调，而且不同接口在实现方式上也不统一，比如有些是基于 Soap 的 Web Service，有些是 RESTfulAPI 等，另外通过这样一个平台可以降低工具之间的耦合度。

举个例子：我有一个应用，调用了很多其他服务，其他服务有时会发生异常，为此我又不想花太多精力去实现与维护。我希望：

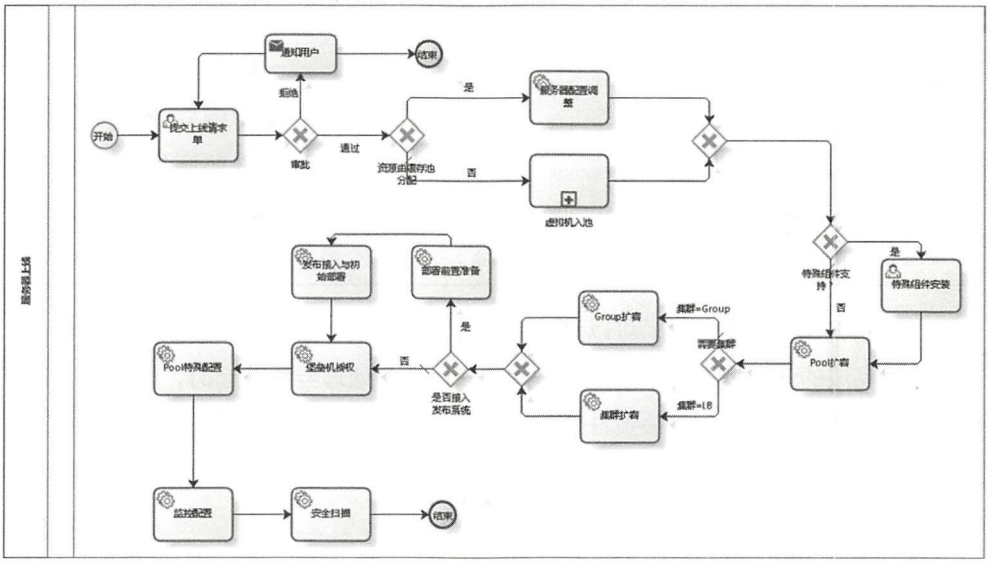
- 及时通知我。
- 及时通知服务提供人。
- 能看到异常发生频率、次数、时间。
- 能看到异常的内容。
- 能看到所有通知事件。

那么，平台是如何运作一个消息从产生到工具的传递呢？

- 首先需要在平台注册一个事件。
- 工具访问之前需要在平台上搜索该事件到并且订阅，需要提供一些必要的信息，比如工具服务的调用地址。
- ENP 根据规则对消息进行格式化并封装。
- ENP 转发消息到外部工具。
- 工具回写消息到 ENP，ENP 最终回写到流程平台。

4．流程场景：服务器上线

下面我以服务器上线流程具体场景为例，进行说明如何设计流程：在经过前期与各个业务部门、运维团队一起分析设计出了最终服务器上线流程图，流程由一系列流对象组成，这些流对象可以是任务，也可以是子流程，比如下图所示的子流程“虚拟机入池”，同时这些任务与子流程之间由连接对象衔接，比如顺序、并行、分支判断处理。流程还需要设定一系列的业务规则，包括审批、状态迁移、SLA、CMDB 数据落地等。



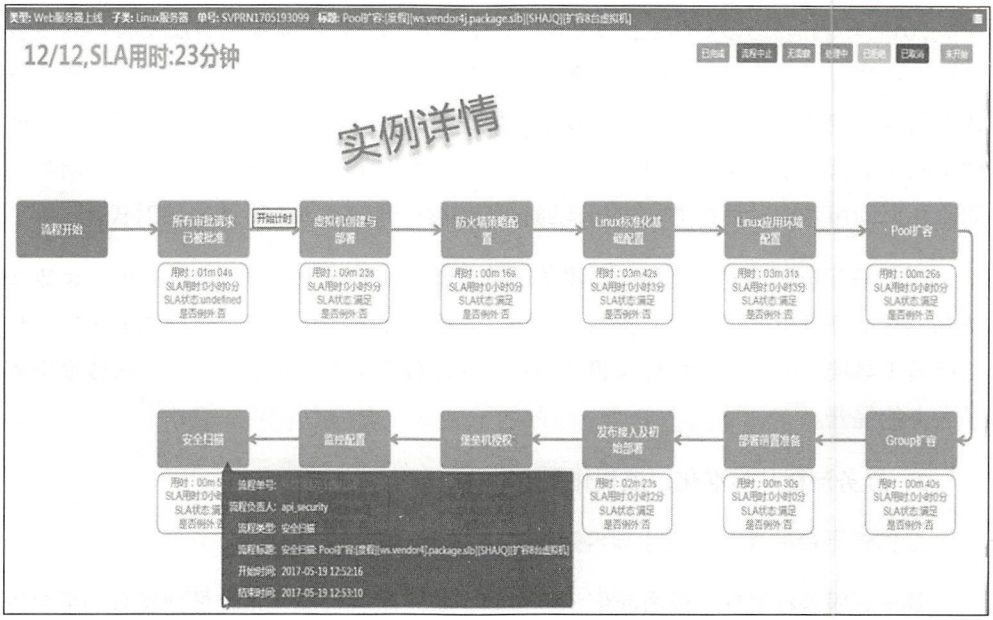
5．可视化的流程运行实例

在有了流程之后，对于不同角色的用户，比如运维人员、开发人员、流程组以及管理人员，需要有个可视化的界面。

- 能清晰直观地看到所有运行的流程。
- 能清楚了解当前某个流程运行或者在哪一个环节等待。
- 流程的运行状态、时效。

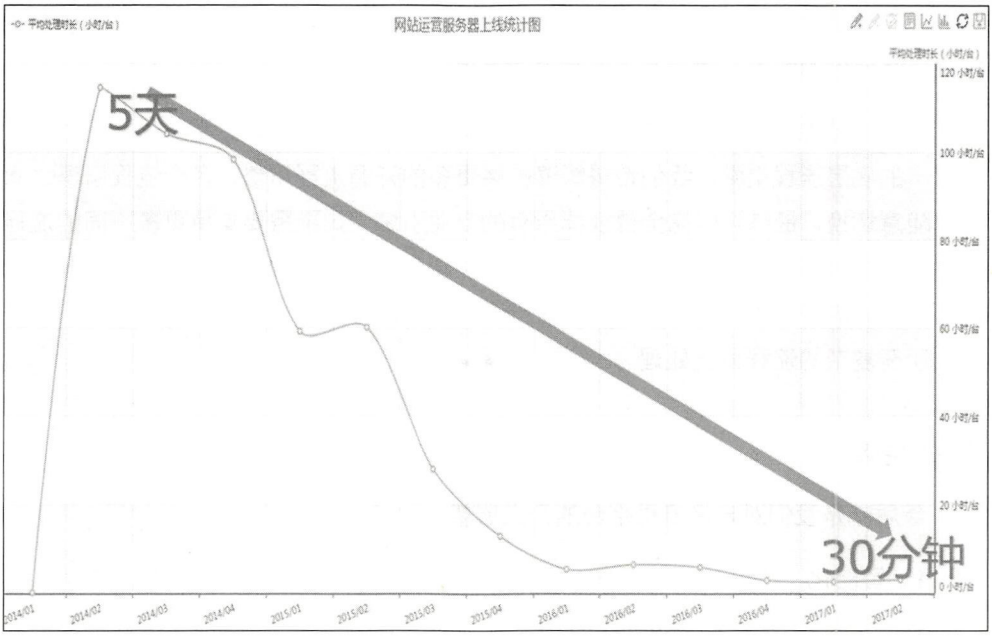
下图就是流程平台可视化界面，可以看到目前所有运行中的流程实例，它们的运行状况，流程的运行时间，不同流程的运行数。当某个流程实例运行超过预定 SLA 时长，也可以查找超时子单，并找到相应的责任团队。

通过打开单条实例数据，可以进一步查看实例详情，在此可进一步看到所有运行或者已经完成的任务以及它们的处理时间。一旦任务处理异常挂起时，可以通过查看任务的责任团队迅速找到相关责任人进行快速处理。



收获总结

1．10X 服务交付能力的提升



2014 年当时还没服务器上线流程时，业务部门从提出需求到最后交付平均需要一周左右的时间，最长可能会达到两周，所有上线的相关环节的协同操作几乎都由低效率的人工完成，很少有工具参与自动化处理。有一个场景让我至今印象深刻，当时整个网站运营中心的各个运维团队座位比较近，所以每当上线的时候，经常会听到不同团队之间用隔空对话的方式进行沟通，还停留在通信基本靠吼的原始时代。

现在有了流程支持，这种低效率的团队合作方式也大大得到了改善，各个运维团队开始尝试开发自己的工具并接入到流程，流程也慢慢地从部分工具接入的半自动化到所有工具接入的全自动化处理模式运行，在流程完全自动化后，最终上线效率得到了大大的提升。

2．服务流程从标准化、流程化到自动化

工作流平台的建设过程主要经过了以下几个阶段的发展。

第一，服务标准化。前面我们已经讲到了关于服务标准化，就是把现有的平台进一步分层，建立标准接口，像 OSG 这种的标准化接口网关。

第二，业务流程化。梳理各项业务，把大部分的 IT 运维工作流程化，确保这些工作都可重复。

第三，流程自动化。把运维人员从低效率、高强度、易犯错的人工操作彻底解救出来，让他们的能力与精力有更大程度的发挥。

在经过成熟期后，现有的流程平台也逐渐暴露出一些问题，最主要的就是过度依赖单一的底层流程引擎。我们希望能够扩展更多的开源流程引擎，多个流程引擎之间可以随意切换，最终可以完全替换掉原有的商业引擎，如果需要支持更多不同的流程引擎，那么对流程进行抽象是必不可少的，并建立出一个新的流程模型。另外一个问题：原来的流程引擎只能处理串行、并行以及简单的分支合流，新一代的引擎可以覆盖到所有复杂的流程分支处理。

未来的展望

最后就是我们对未来流程平台的几点展望：

1．智能化

目前告警种类过多，很多流程上的异常告警在经过人工分析后，多数是不需要处

理的。我们的目标是系统能做一些基础的分析，并在自动诊断问题后，对问题进行自我恢复，对于有些需要人工介入的，也能做些初步的分析，同时把经过分析后的日志发送给处理人员，这样可以提高处理人员的效率。

2．自助式的流程编排

目前流程的开发还需要流程团队参与，我们期望在未来，所有的用户可以自行编排流程，自行定义流程中的每一步工艺，所有的工艺可以发布到一个共享的市场，当用户需要编排流程的时候，除定义自己的工艺外，也可以引用其他团队已经写好的工艺。

5.2.2　SWAT 团队，排障有我

网站运营是一个非常复杂的过程，网站的稳定性对一个互联网企业来说，尤其重要。携程线上业务众多，每个应用都有较高的可用性要求，如何在出现故障后，能快速地恢复网站运营，对所有网站运维人员来讲无疑是一个极大的挑战。

一个网站的稳定运行，其中有软件和硬件的支撑，也有工具、人员、流程的有力保障。任何环境的故障或者失误都有可能影响网站运行。比如，机房某台服务器温度过高自动重启，工具的 Bug 导致过多占用系统资源，发布人员失误发布了不正确的版本等。

故障并不可怕，关键是能否在最短的时间恢复！我们的网站运营中心，根据长期的运维经验，总结出一套故障快速处理方法，那就是 SWAT 团队机制。

SWAT 团队的定位和职责

具体的组织方式是这样的：SWAT 团队按产品线成立小组，抽调产品线最精锐的开发人员和运维技术工程师加入，成立虚拟团队，主要职责是负责一线生产故障原因的快速定位和问题解决。另外团队成员必行 7×24×365 待命，一旦收到网站监控中心的故障通知电话，必须第一时间加入电话会议，与网站运营中心负责人一起处理故障。

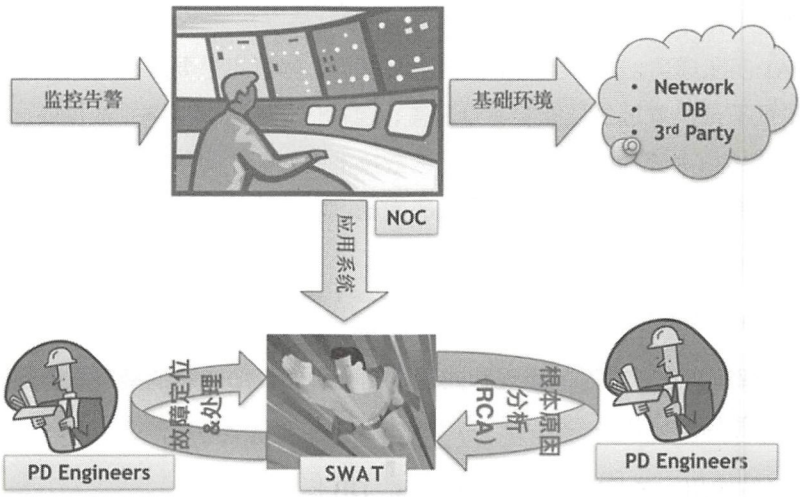
SWAT 团队的职责主要有以下 3 个方面：

- 故障期间，定位和处理网站应用系统异常所引发的高优先级事件。
- 故障恢复后，主导 RCA /COE 分析，分析故障的根本原因，提供技术改进方案。
- 日常期间，关注应用性能问题，解决网站潜在隐患。

SWAT 的工作流程

接下来介绍一个具体的 SWAT 团队处理故障的例子。

2017 年的一天深夜，网站监控中心突然发现某业务生产订单异常跌零，怀疑有生产故障。监控中心的工作人员立即开启电话会议，呼入对应业务订单系统的开发 SWAT 成员 A 以及运维 SWAT 成员 B，二人电话会议短暂沟通，发现表面没有做任何生成变更，排除是发布变更引起的故障。接下来成员 A 发现应用的日志大量读取缓存数据造成超时错误，定位到是访问缓存数控响应时间变长，导致大量超时，怀疑是缓存服务有问题。接下来网站运营中心呼叫缓存服务的 SWAT 成员加入会议，继续排查缓存服务问题，定位到是网络问题。最后，呼入基础运维 SWAT 人员加入电话会议排障，告知此时此刻正有大量基础服务维护，过多占用网络资源，停止基础服务维护，故障解决。



整个故障处理过程非常迅速，每个呼入对象都是对应服务的负责人或技术一把手，都能比较全面地了解系统，做到第一时间定位问题，快速回复网站。

故障处理完了还没有完，接下来是反思与总结。SWAT 团队成员要能主动牵头对该故障进行根本原因分析，刨根究底找到问题发生的源头，以及后续的改进意见。寻找导致故障发生在人、工具、流程方面的不足。比如，根据 SWAT 成员牵头，召开了故障复盘与分析会议，最终确定有以下几个方面的不足。

1．人员方面。没有对生产环境集成服务维护引起足够的重视，导致变更比较随意，以至于引起业务影响也浑然不知，这是人员安全意识薄弱，需要加强教育。

2．工具方面。基础工具不健壮，没有对其维护期间产生的网络压力进行合理限制，直接导致业务流量的争抢，引起故障发生。

3．流程方面。维护工作未遵循变更流程，基础维护也是变更，一旦故障，影响面有时候非常大，说明执行过程中没有严格执行变更规范，导致系统无变更记录，故障定位时间比较长。

以上方面的改进，都会有相应的负责人负责整改，以及在相应的时间完成整改。最后还要提的是，SWAT 团队不只是在故障时第一时间冲锋在最前线，平时也是系统安全隐患和系统瓶颈的巡视员。他们时常巡视系统的监控、告警，实时运行数据，发现一些系统潜在的问题时，提出改进意见，尽最大限度避免生产故障的发生。

SWAT 团队需要的知识与技能

SWAT 团队成员，通常要具备技术全栈能力，当然不是要求每个人都是专家，至少 SWAT 所有成员掌握的技能要尽量覆盖，包括应用、公共服务（数据库、缓存、消息）、基础架构（服务器、网络），有了这些技术能力，才能第一时间得到最权威的问题分析与排除方法，争取在第一时间解决问题。以下简单总结了需要具备的一些技术能力的方面。

1．了解应用架构和部署架构。

2．了解应用在异常时的表现和处理机制（例如，所依赖的接口服务超时或不可访问时会有怎样的症状）。

3．了解应用间依赖关系（包括 Cache、DB）。

4．了解应用核心逻辑的实现方法。

5．深厚的编码功底，对多线程、Socket、Caching、GC 机制、异常处理等有充分经验。

6．深入理解.NET Framework/IIS 原理和运行机制。

7．精通 HTTP/TCP 协议知识。

8．精通 HTML/JavaScript 知识。

9．相关的数据库和 SQL 知识。

10．熟练掌握各种监控工具，能快速获取排障所需要的信息。

11．日志分析能力（IIS Log、Windows Event、Clogging）。

12．DUMP 分析能力。

13．HTTP/TCP 协议分析能力。

故障发现与识别

另外，光有技术还不行，SWAT 团队还要能识别与定位故障。能利用一些工具，快速定位发生自不同层面的故障和问题。

业务层方面，要能了解订单量告警意味着什么，用户注册与登录数增多又意味着什么，什么情况下是故障，以及判定时态的危机程度等。

应用层方面，要能理解和知道这些指标背后意味着什么。比如，服务请求数、URL 响应时间、Clogging 中异常数量、集群成员服务器健康状态、DB Block 等，了解这些信息就知道到底会影响哪些业务模块。

系统层面，比如：High CPU、Memory Leak、Connection Stacking、Network Traffic overload 等对业务的影响有哪些，如何快速恢复等。

只有识别这些指标和信息，才能快速直接定位问题，尽可能缩短故障恢复时间。

总结：SWAT 团队与机制已经成为网站运营中心抵御系统故障最有利的手段。有了他们的存在，才能有力保障系统的稳定性。本质上也是 DevOps 的实践，一支团队协作的体现，个人主人翁意识的体现。

5.3　沟通有术

软件产品本身是沟通的产物。从客户需求的前期调研到需求分析、架构设计、编码设计实现、贯穿项目过程中的测试修改、上线发布等这一系列工作都是以沟通为基础的。沟通不仅仅是软件项目管理的必要手段，沟通更是软件项目建设过程中的必需工具和必不可少的重要工序。

当下，不仅仅要解决人和人的沟通问题，还要考虑人和工具、人和外部系统的沟通问题。如何以快速、自助化的方式高效获取有用的信息，最快速度达成共识，这将是敏捷工具面临解决的问题。接下来谈谈携程技术人都有哪些常用的沟通技术和方法。

5.3.1　DevOps，开发和运维终于在一起了

近年来，DevOps 越来越火，其热度一点不亚于 Cloud、BigData、AI 、VR 等，各种各样的 DevOps 技术研讨会，Meetup 活动信息一直刷屏着我们的朋友圈，那到底什么是 DevOps 呢？

要给 DevOps 下个简明、准确而又恰当的定义真不是件容易的事。不过，以前看到过一句话，似乎能较好地解释什么是 DevOps：“DevOps 是一种文化、运动或者实践，它强调软件开发人员和其他 IT 专业技术人员之间的沟通与协作，共同促进软件交付流程和基础设施变更的自动化，最终实现用户价值的快速交付。”

为什么需要 DevOps

先来讲讲为什么需要 DevOps？随着软件开发过程的分工、产品定义、需求分析、开发、测试、发布、维护、运营等环节逐步变为专人专事，分工本身是没有问题的，通过采取分治策略，处理复杂问题。只不过，人都是懒惰的，随着时间推移，人与人之间开始有了隔膜，各自岗位的人员不再了解彼此的工作内容，各行其是，工作内容和背景开始互相不理解，各自退守自己的地盘。比如，开发人员为了业绩，只顾手头开发任务完成，尽量上线新功能，根据经验我们也知道，系统只要有代码修改就有可能带来不稳定的因素，这个时候往往运维人员会希望系统尽量稳定，能不做变更尽量不做，这样就形成了矛盾，开发和运维人员开始有了隔膜。

解决的问题

下面是一个大家都基本熟悉的例子：部署软件产品。

开发部门要开发一款新产品，这款产品要使用最新、最炫的技术来保证客户所有花俏的需求，从而给公司带来数百万元的利润。这款产品被要求使用最新的技术和运行平台，还得马上交付。于是开发部门没日没夜的加班、赶代码（cuts code like crazy），终于如期完成了任务。然后他们把自己的“杰作”一股脑地甩给了运维部门，后者还没能完全接手，前者已经迫不及待地开始了庆功会。接到产品后，运维部门每个人的心中都充满了恐惧。

下面就是运维部门的恐惧之源：

- 这款优秀的产品在目前的底层平台上无法运行，因为这个平台太古老了，空间不足，不支持某某版本。
- 这款产品的体系结构与我们的存储、网络、部署、安全等模型不匹配。
- 这款产品的报告、安全、监视、备份、服务提供我们搞不懂 ，所以没法把它做成实际可用的产品。

尽管伴随着不绝于耳的抱怨和咒骂，运维部门最终还是把这款产品安装好了。不

幸的是，由于做了很多蹩脚的修改和不合理的强迫式运行，这款产品的性能大打折扣，最终这个项目被定义为失败。

于是非常沮丧的运维部门开始记录各种问题，源源不断地给开发部门提 Issue。而开发部门的回应基本上都是：

- 这不是我们的错，我们的代码非常完美，是运维部门的部署做得太差劲了。
- 运维部门比较笨，他们不懂新技术，为什么他们没法实现最新的技术呢？为什么他们这么落伍呢？
- 在我的机器上运行没问题啊……

两个部门之间的交流很快变成了一场暴风骤雨。客户以及股东、投资方和管理层则成了蒙受损失的失败方。最终公司损失了无数的金钱，大家也都失业了。

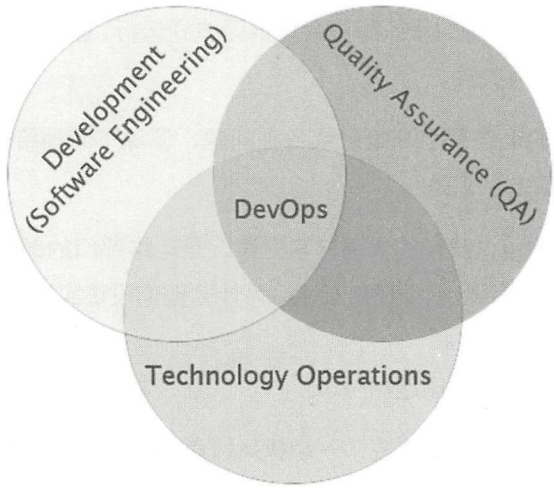
本质上，DevOps 是解决团队协作的问题。通过 Dev 与 Ops 的密切配合，各自贡献自己的技能。其中 Dev 更擅长创造，Ops 擅长运营，彼此拥有共同的目标，积极承担相同的产品按时、保质交付的职责，遇事不埋怨、不推诿，形成高效协作的工作团队，最终实现用户价值的快速交付。

DevOps 的开发方法

DevOps 的开发方法是敏捷（Agile）以及精益（Lean）开发概念的延伸。有别于传统开发流程，DevOps 的开发方法打破了每个独立的阶段，从需求分析、系统设计、程序开发、安装测试、后续维护再回到第一阶段，形成一个封闭循环。

DevOps 的开发方法要求开发人员持续改善并整合不同的阶段，加以组织过去任务所发生的事，开发人员需要自动化工作流程。自动化开发周期的每一个阶段，不仅需要自动化测试，还要自动化部署，并且提供自动化的数据给所有参与的人，让所有人可以合作。

过去的文化需要改变，现在必须量化每件事，依照量化的数据，改善每个阶段，这些是根本性的改变，企业主可监看过程产生的数字而不仅是是生意上的结果，才可以用更快的速度创新，更快地抓住市场份额，让顾客开心，以上都是 DevOps 开发方法的关键改变。



打破沟通壁垒

1．角色之间

尽量做到一专多能。一个完整的研发团队人员，往往包括产品设计师、需求分析员、开发工程师、架构师、测试人员、运维工程师、运营人员，编制完整或者一人身兼数职。比如，担任 Scrum 角色的产品经理负责产品设计，架构、开发工作往往也由同一个人担任。这里团队成员就有个疑惑，到底我们团队是健康的还是不健康的呢？我做的工作是专业的吗？尤其是身兼数职的会更加怀疑自己现在的角色设置是否合适。其实不然，一专多能，恰恰是一个团队降低沟通成本的最好方法。举一个例子，往往我们遇到能力比较强的开发人员，有时候“单兵作战能力”能以一抵十，为什么呢？当然一方面是其能力的确很强，另一方面是省去了很多做不同类别工作的沟通成本。如果一个人同时具备需求的分析、开发和测试能力，其实就省去了和需求分析人员、测试人员的沟通成本，一旦有需求，二话不说就开干，写代码、跑用例一气呵成，如果还能知道如何运维线上环境，那就更好了。这就是为什么说 DevOps 时代，我们鼓励要做全栈工程师的原因。当然，有时候也不能让一个团队人员过少，出于团队风险的考虑，有可能某个核心人员就是单点，项目容易受到个别人员状态的影响，这也是需要平衡的问题。

2．部门之间关系

早参与，多参与。对于开发人员，要让运维人员常驻开发部门，全程参与开发流

程。邀请运维人员参与你的 Scrum 或者开发会议，与他们分享项目计划、新技术的点子和心得。收集功能性需求（指开发人员用到的需求）的同时也要收集运维方面的需求。把对于发布、备份、监控、安全、配置管理和系统功能的测试作为一项独立的项目流程。软件产品在开发时解决的问题越多，那么在使用时暴露给用户的问题就越少。给运维人员做培训，让他们弄清楚项目的体系结构和核心代码。如果运维人员在反馈 Bug 时提供的信息越多，那么你花在排查问题（trouble-shooting）的时间就越少，这个 Bug 也就会更快地被解决。

对于运维人员，在遇到问题时需要把开发人员加进来，大家一起解决问题。邀请开发人员参与你们的会议，分享项目进度（roadmaps），并且共同修订工作计划。运维人员一定要了解开发部门下一步的工作方向，从而确保产品运行的底层平台能够良好地支持最新技术。开发人员也会带来相关的技术、知识和工作，帮助你们改善产品的运行环境，使其更加易于维护、有效。

有一些开发领域的概念，例如：要根据 API 而非封闭的 interface 来构建工具，分布式版本控制，驱动测试开发，以及诸如敏捷开发、看板管理（Kanban）和 Scrum 等方法论。如果把这些概念应用在运维领域，同样会产生革命性的变革。

不要惧怕新点子和新技术。我们可以随时随地向他人学习，哪怕是一句“我们再也不要那样做了！”也会让我们从中获益。尽管处于不同的部门，但是我们要共同学习、共同成长，这样才能更好地协同工作！

按照从高到低的顺序，有效的沟通方式应该是：面对面交流、视频会议、电话、即时通讯软件、Email。

自动化一切

自动化，自动化，自动化，重要的事情说三遍！在软件交付整个生命周期过程中，涉及诸多任务，很多工作是需要不断重复的。比如，编译、打包、单元测试、集成、API 测试、性能测试、部署、用户反馈收集等。往往这些工作看似很简单，不会占用过多时间，但是日积月累，这个总体时间是非常恐怖的。另外，人工重复执行，很难保证应用环境一致性，也就是说，人不可能不犯错误，但是往往遇到一次错误，就会给系统可用性带来不必要的损失。所以，我们要卯足“自动化一切”的劲儿，来完善整个研发流程。

工具集

使用 DevOps 工具包中合适的工具，可以帮助我们很好地实施 DevOps，优化敏捷发布过程和增强团队协作。先声明，DevOps 不仅涉及工具，如果背后没有合适的人员与文化，即使拥有最好的工具，也不能成功实施 DevOps。不幸的是，没有“文化”工具可供你使用，让你能够立刻在团队之间培养协作和反馈。

合适的工具可以提供框架，帮助公司成功实施 DevOps。你选择的工具，应该鼓励反馈，并防止进一步形成孤立。工具还应该帮助统一和协调团队。确定采用的 DevOps 工具包是成功实现 DevOps 目标和量化的关键第一步。虽然工具的特性集和解决方案是很重要的，也要重视工具组合起来的效果。无法整合的工具可能会需要过多的维护、成本，或产生冲突的信息。

在一个非常简化的应用生命周期视图中，笔者将过程划分为四个主要步骤：规划，设计，部署和维护。在每一个步骤中，都有可以增强这一环节的工具。同样重要的是，这个过程不是一次性的，是一个持续的循环，这种持续的反馈周期是 DevOps 成功的必要基础。

与其通过一系列的产品列表来选择 DevOps 工具，不如你应该考虑自己的应用生命周期，根据特定的目标来做选择。

阶　段	分　类	工　具	解决问题
基础	云/基础设施	Azure AWS Rackspace Joyent CloudFoundry	快速构建基础资源，按需获取
	虚拟化工具	KVM Xen VirtualBox Docker	快速构建资源池
	应用智能	AppDynamics OneAPM Hubot	人和工具协作

续表

阶　段	分　类	工　具	解决问题
规划	数据库	MongoDB Cassandra hBase MySQL PostgreSQL Redis	结构化数据
	搜索	Solr ElasticSearch	知识库
	Web 服务器	NGINX Apache	业务表现与接口
设计与架构	扩展	ActiveMQ RabbitMQ Memcached	队列服务
部署	容器	Docker Kubernetes Mesos	容器编排
	持续集成	Jenkins	一致性，降低风险
	配置管理	Puppet Chef Ansible SaltStack	环境标准化
维护	告警	OneAlert PagerDuty ServiceNow VictorOps BigPanda	故障发现
	日志记录	Splunk SumoLogic Loggly Logentries ELK	问题回溯

DevOps 能力度量

下面为用来检验你的组织对 DevOps 应用情况的清单。

- 开发团队和运维团队之间没有障碍，两者皆是 DevOps 统一流程的一部分。
- 从一个团队流到另一个团队的工作都能得到高质量的验证。
- 工作没有堆积，所有的瓶颈都已经被处理好。
- 开发团队没有占用运维团队的时间，因为部署和维护都处于同一个时间盒里。
- 开发团队不会在周五下午 5 点后把代码交付进行部署，剩下运维团队周末加班加点来给他们擦屁股。
- 开发环境标准化，运维人员可以很容易将其扩展并进行部署。
- 开发团队可以找到合适的方式交付新版本，并且运维团队可以轻易地进行部署。
- 每个团队之间的通信线路都很明确。
- 所有的团队成员都有时间去为改善系统进行试验和实践。
- 常规性地引入（或者模拟）缺陷到系统中，并得到处理。每次学习到的经验都应该文档化并分享给相关人员。事故处理成为日常工作的一部分，并且处理方式是已知的。

DevOps 实施的契机

具备以下几方面的因素，可能促使一个组织引入 DevOps 实践。

- 使用敏捷或其他软件开发过程和方法，团队开始具备基本流程优化意识。
- 业务负责人要求加快产品交付的速率，甚至要求当天交付。
- 虚拟化和云计算基础设施的使用日益普遍。
- 数据中心自动化技术和配置管理工具应用的普及。
- 传统的跨职的研发组织架构与管理方法，造成开发与运营人员之间的隔膜，因此需要 DevOps 能力来克服由此引发的问题。

DevOps 实践总结

DevOps 的目的是打造持续增量的价值流并杜绝浪费。任何不以消除浪费为目的的 DevOps 实践都是假的 DevOps。我们实践 DevOps 的目的是实现从一个想法到真正把这个想法形成产品、服务，并提供给用户去用的流程。缩短这个流程的时间，提高

这个流程的效率是 DevOps 实践的一个最重要的目的。我们要让每一个步骤，每一个过程的价值都是递增的，而不是说产生等待，或者说产生依赖。比如对配置管理的依赖，对人员的依赖，这都是有悖于这个目的的。所以我把这句话送给大家："DevOps 的目的，最重要的一点就是加速高质量的交付，提升用户价值。"

在实践过程中，我们可以将各个子环节的自动化流程作为一个起点，很多时候你可能没办法一次性把整个部署流水线构建得那么完美，我们就可以分析是不是在每个子环节里面已经实现了足够程度的自动化。比如说自动化发布，现在是不是还要靠人工去操作，这些是最基本的动作，做好这些之后，你才有能力或者有条件和上下游进行对接。只有完成了每个环节的自动化后，我们才有可能去构建整个部署流水线。也就是说我们在落地的时候可以想想整个业务流程里面的痛点，比如说是你的部署没有完成自动化，还是测试没有完成自动化，导致了这个流水线没有办法流传下去。然后以每个环节的自动化作为一个开始，然后把它们集成起来，就可以实现整个价值流的快速交付。

5.3.2　ChatOps，技术团队新的沟通方式

何为 ChatOps

说到 ChatOps，自然会想到 DevOps。软件开发领域这两年"疯狂"地传递着由 Dev 和 Ops 向 DevOps 转变的理念，DevOps 的核心是什么？有哪些实现方式？过程中又存在什么问题？

DevOps 包含四大核心：技术、组织、流程、文化。以自助、自动为指导思想，具体 DevOps 落地可以从 CI/CD 着手。DevOps 落地很难，一般情况下，因为有太多历史债务，有太多规章制度约束，在实践落地过程中无非就是玩玩工具，个别运维人员写写脚本。DevOps 尚且如此，那看起来更高大上的 ChatOps ，我们应该如何正确看待呢？机器人？聊天室？机器人聊天运营？先看看 *SneakyCode* 上的总结："At the heart of DevOps is CAMS …… ChatOps is an extension of DevOps and enhances it with and extreme focus on CAMS"。这里，CAMS 是指 a culture of automation，measurement and sharing，认为 ChatOps 是对 DevOps 的一个实现与加强。

一直以来，运维人员好的工作方式给大家的感觉就是脚本，部署要执行脚本、变

更要执行脚本。或者从进阶一层来看，运维会用各种小工具，比如 Puppet、SaltStack 等，对脚本形成统一管理、下发、执行。很多人都在讲，要把繁重且重复的劳动交给机器人，让人做更有趣、更创新的事情。比如运维人员所做的日常巡检、故障处理，可以由这些机器人来协助处理。而作为运维人员的机器人，一个参与工作的很好方式是加入到我们的日常聊天组，一起共事、一起学习。这样的工作方式听起来是不是很酷?

我们的运维技术团队，密切关注当下前沿的技术动向，业界有好的工具和工程实践都会第一时间在公司内率先试用，好的方式会逐步推广。

运维工作的日常沟通

在过去的时间里，我们运维团队和开发团队的日常运维沟通（如新上线一台服务器，备份运行中的数据等）往往都是通过会议确定方案，发送邮件来确认需求，在维护过程中，还会通过电话或者即时聊天工具（如微软 Lync）来进行及时沟通，确保执行过程中不会出现偏差。其中变更管理也有对应的工具，比如，我们有一个非常强大的流程管理系统 remedy，执行变更期间，各个过程的数据和状态都在这个系统中统一管理，但是在执行过程中难免也会出现状况，是工具不能及时处理的，由于沟通不及时，导致失败也是常有的事。比如：线上进行新代码发布，由于不知道基础服务团队正在做网络变更，导致发布进行过程中很多网络中断干扰被误认为是代码问题或者服务器自身问题，故障排查一开始就走错了方向，浪费了时间，影响了生产应用可用性。传统的沟通方式有以下特点。

1．面对面沟通。需要聚集相应人员，面对面沟通能直接获取彼此反馈，效果好，但是内容不可回溯。

2．电话沟通。突破地理限制，不能面对面沟通，实时获取反馈，内容不可回溯。

3．邮件沟通。突破地理和时间限制，不能面对面沟通，通常反馈比较慢，可以过时查看内容，内容可以回溯。

4．聊天工具沟通。突破地理限制，可以及时沟通，也突破了时间限制沟通，除了一些比较弱的 IRC 工具（如 Lync 等），内容通常可以回溯。

实际工作中，采取上述任何一种工具都不能很好地处理技术运维过程的沟通问

题。比如，处理故障过程中往往需要配合运维工具辅助排障，查看系统应用日志，监控告警数据等。及时通知相应系统 Owner 及时参加排障碍，争取获得更多的帮助等，一旦故障原因确定需要及时进行故障恢复，批量执行一些繁琐的操作等。这个处理过程都是分秒必争的，其中信息传递的及时性和准确性尤其重要，同时也要保证执行操作规范、可靠，过程可回溯。

机器人可以是好帮手

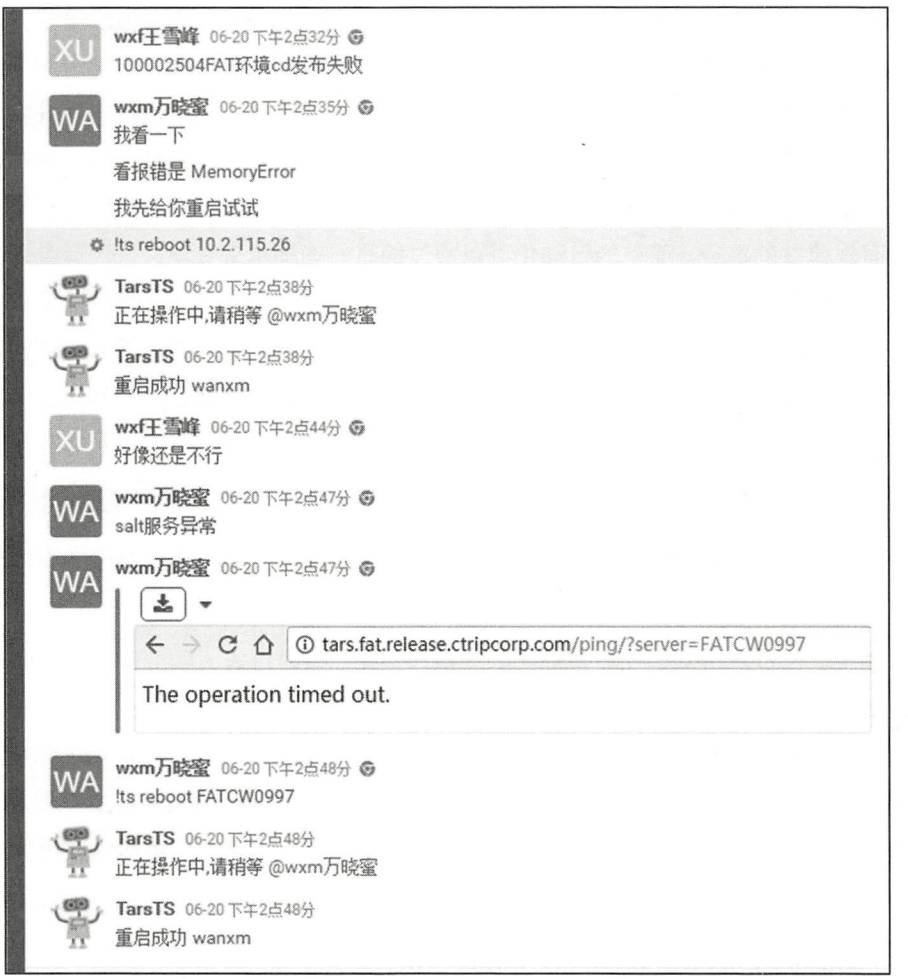
现在工程师的人力成本越来越高，靠工程师手动通过 SSH 或类似的第三方工具来远程部署服务的方式，枯燥乏味，容易出错，部署时间长。仅仅为了部署服务，工程师就免不了加班，而且整个过程都要守候在工作机器旁边，片刻也不能离开。老板不愿看到员工既加班又不能保证服务质量，工程师也不想要这样重复而紧张的部署过程，希望能够找到一种快捷安全的部署服务的方式。众所周知，机器比较擅长重复性的、有规律性的事务，简单的自动化可以通过编写一些脚本来实现，替代运维人员繁重的手工操作。这样，一方面提升了运维人员的工作效率，另一方面也保证了工作流程的一致性，避免了人为因素导致过程差异。当然，机器能做的事情不只是这些，还可以赋予更多的能力，具备一定智能的运维代码，我们说它就是运维机器人。

我们的日常运维同样经历过以上描述的场景，后来我们就在思考，如何换一种工作方式，让自己既能提高技能，也能节省时间，更重要的是，让服务更加稳定可靠。即将成为趋势的 ChatOps（即一种会话驱动型开发的做法）的工作方式不失为一种好选择。

首先，让自己日常工作能自动化的尽量自动化。当前机器人的发展响应人们的需求，本质上还是基于命令方式的，需要打造一套自己的运维任务库。比如，以云平台运维机器人为例，通过一个命令查看云平台集群监控状况，通过一个命令灰度升级一个集群，通过一个命令下线维护一个宿主机等。这些都是事先编写好自动化脚本，完成测试后加入任务库，后续给运维机器人调用执行。然后打造自己的运维机器人，现在市场上 ChatOps bo 的开源实现呈三足鼎立之势。

- Hubot：CoffeeScript 实现，GitHub 提供且自用。
- Lita：Ruby 实现，支持容器部署，依赖 redis。
- Err：Python 实现，笔者目前还没有用过。

我们的运维机器人是基于 Hubot 打造的，Hobot 是 GitHub 在多年前开发的一套用于管理 GitHub 自己的软硬件的机器人。中间经历了自用、开源、重写再开源三个阶段，现在俨然成为 GitHub 上最火热的项目之一。该项目社区相当活跃，GitHub 上有数以千计的各类自动化 plugin，如果要搭建一个自己的机器人，分分钟就能实现。当然，要能利用自己的环境，帮助自己提升运维效率，还要因地制宜，开发自己的自动化插件。我们相继开发了云平台运维机器人、NOC 运维机器人、发布支持机器人等富有特色的机器人。



打造群聊系统

过去，在我们内部，大家通过微软提供的 Lync 进行及时聊天，很大程度上解决了我们内部各类人员跨地理及时交流的问题。通过 Lync 用于日常交流到是基本满足需求，但是也存在一些问题。比如，聊天记录不能保存，聊天结束自动清理，不方便回溯；创建群聊比较麻烦，不能很好地保存群聊背景信息，第 3 个人加入也不能看到之前的聊天内容，需要群主 copy-pase 聊天内容才能实现信息传递；还有消息格式单一，不支持一键格式化的内容（如 Markdown 等），表现形式有限。这些不足，在技术人员进行技术排障沟通时尤其突出，我们的技术团队迫切需要一款软件解决上述问题。

CtripTeam 是我们运维技术团队打造的一款及时聊天工具，该工具能很好地解决上述 Lync 面临的问题。使用该工具能很好地实现点对点沟通、认证授权和公司 SSO 系统集成；能进行公司同事通讯录快速查找；可快速创建群聊，且聊天内容永久保留；可以方便内容回朔，再也不用为后加入群聊的同事 copy-pase 聊天内容了；部分聊天频道和运维机器人对接，机器人用特殊角色加入群聊，发布一些事件信息，响应系统管理员发出的指令等。

现在，CtripTeam 在日常运维工作中的重要性越来越大，我们每天可以不打开邮件软件，但是不能不打开 CtripTeam ，它已经不是单纯的聊天工具，它还是一个信息发布平台，我们的日常工作也有大部分是通过它去驱动机器人完成的。

工具平台化，服务化是前提

我们运维团队能以很快的速度，在比较短的时间里由 DevOps 过渡到 ChatOps，与之前极力倡导的 Ops 工具平台化、服务化密不可分。基础平台管理的工具有：数据中心管理系统 IDC Manger，自动化数据中心物理设备管理，比如物理服务器、机柜、电源上下线等；网络设备管理系统，Netanger 负责各类网络设备自动化管理和监控；我们的数据中心操作系统——CDOS ，统一对数据中心各类资源池化（虚拟化），进行资源统一调度和分配；另外，还有监控系统、日志系统、告警系统、发布系统等。这些工具都有对外提供服务，提供统一的 API，能被第三方系统调用。有了这些作为基础，再来打造 ChatOps 就容易多了，只要把这些工具和数据有机地组织起来，解决信息聚合和流程串联问题，就能进一步实现效率的提升。

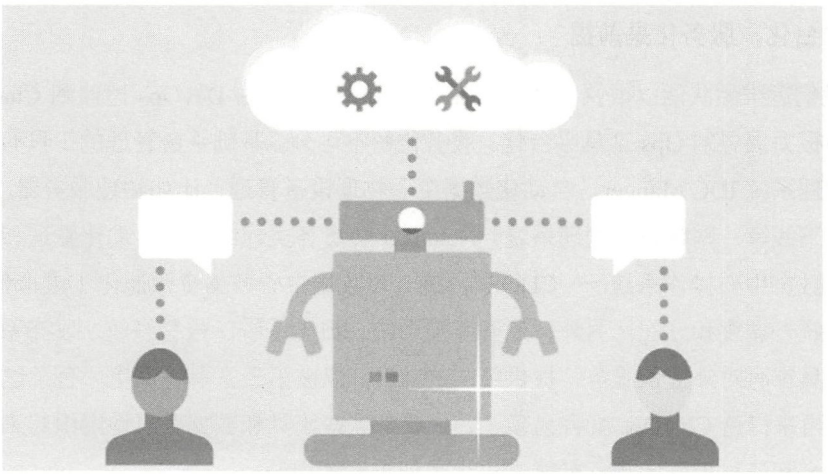
人，机器人，工具沟通的革命

ChatOps 将运维人员、工具、机器人有机地联系起来，通过聊天群或者信息频道，实时接收运维工具系统和运维人员的工作状态信息。实现了信息的高效传播和任务的状态透明呈现，缩短了问题反馈周期，最终增强了团队协作，提升了工作效率。

消息的汇聚，运维人员不再需要主动地去各个系统来回切换获取关联信息，各类系统的数据能够被机器人主动推送，呈现到运维人员的眼前。比如，在系统发布升级的过程，如果发布失败，机器人会主动推送发布失败日志摘要和详细内容地址链接，主动推送当前生产订单影响情况，发送订单趋势图，给出大致问题诊断建议等。这一切都在独立的应用发布频道呈现，省去了寻找信息的时间。

快速执行指令，运维人员不再需要记住一些常用的运维指令，通过自然语言和机器人交流就是实现一些高级操作。比如，以发布升级过程为例，如果发布失败，根据汇聚的信息发现是有一台机器中的服务器死机了，需要替换一台新机器，完全可以直接告诉机器人扩容一台新的机器，并将死机的服务拉出集群，不再需要人主动登录各个系统完成资源申请与集群设置等，整个过程一气呵成，省时省力。

过程透明，可回溯，运维人员再也不用害怕因为过程不可终结而烦恼，运维过程都是透明的，聊天室的信息就是日志，过程完全可以回溯。尤其在处理生产故障期间，每一个人的发言和执行指令都是可以追溯的，在非常方便的时候进行问题分析和问题改进。



ChatOps 带来的效率提升

ChatOps 站在巨人的肩膀上发展，也为工作带来了显而易见的好处：

- 公开透明。所有的工作消息都在同一个聊天平台中沉淀并公开给所有相关成员，消除沟通壁垒，工作历史有迹可循，团队合作更加顺畅。
- 上下文共享。减少因工作台切换等对消息的截断，保证消息的完整性，让工作承接有序，各角色、各工具都成为完成工作流中的一环，打造真正流畅的工作体验。
- 移动友好。只需要在前台与预设好的机器人对话即可完成与后台工具和系统的交互，在移动环境下无需再与众多复杂的工具直接对接，大大提升了移动办公的可行性。
- DevOps 文化打造。用与机器人对话这种简单的方式降低 DevOps 的接受门槛，让这种自动化办公的理念更容易地扩展到团队的每一个角落。

虽然我们接触 ChatOps 领域时间不长，但已深刻感受了其独特魅力。

1．聊天室不再是聊天，随着伙伴角色的丰富与能力的提升，聊天室会成为一个协作、学习的基础支撑平台。当然，不同于现在很多微课堂，这里会让你看到高手的实操方式，让每个成员可拥有很酷的机器人拍档。

2．生态从小团队做起。以前一说生态就被放得很大，即使企业里面说生态，也时常会放到基线平台的概念里。现在我们真的可以快速建立小生态了，你只要在一些基础上使用一些机器人，就可以为你的团队生态提供一份贡献，相信每个企业的可优化空间都很大吧。

3．合理处理人与机器的关系，不要再是 e-e 关系 ，而把它作为团队里的成员，当作最吃苦耐劳的成员来看待，这才是 ChatOps 所期望的。

4．止于至善，在 IT 这个领域，尤其是现在的 DevOps、ChatOps 领域更为适用，这些都是点滴积累、精益求精的建设过程，不可能有万能钥匙（标准产品）。

ChatOps 代表着“透明”和“效率”的趋势，无疑是团队沟通和合作发展的一种主流方向。首次打通这种工作流程可能需要一定的工作量，但一旦打通并走上正轨，将会为整个团队培养出一种更加简单高效的工作文化。

5.3.3 打造一站式项目管理平台，助力研发效率提升

作为国内领先的在线旅游平台，随着业务迅速发展，研发团队规模也快速扩张。相应的，如何降低项目管理成本，提升研发人员工作效率，保证项目交付质量，变得日益重要。

为此，我们尝试对“需求→开发→发布”整个链条中的各种流程、工具、数据进行打通和自动化处理，以此减少研发过程中的各种浪费，提升整个研发体系的效率。

项目管理之困局

问题 1：信息不一致，沟通成本高

各部门各团队，他们使用的项目管理工具各不相同（Trello、Wiki、Mingle、Excel、Email……）。这些项目信息存放的格式、载体的差异性，导致了对于同一个需求，从不同渠道得到的信息可能是互相矛盾的（如机票团队记录一个需求已经是“测试中”，但支付团队记录则是“开发中”）。

如果想对齐准确的项目信息（如各团队开发进展，计划什么时候联调/提测？），势必要用“当面沟通、打电话、聊天群、邮件”等人工的方式跟每个开发团队确认。沟通效率低不说，还经常会打断开发人员的工作。

同时，互联网的项目模式又强调“小步快跑、快速迭代”。项目的规模小，但项目的数量多。加上低效的项目沟通方式，最终导致了我们整体的项目管理成本极高。当时甚至有些研发经理跟我抱怨，基本白天的时间就是协调各种项目资源、调整计划，只有晚上才有时间写代码。

问题 2：老系统不好用，改造成本高

当时有一套老的项目管理系统，但对于整个研发工作流（需求→计划→开发→测试）并没有设计打通，需要开发人员自己记得在不同阶段到不同的表单列表中填报项目信息。同时各阶段的表单字段异常复杂（以版本提测环节为例，相关表单中有几十个字段，平均每个项目需要花费 30 多分钟完成填写）。研发团队普遍觉得系统不够人性化，所以大家实际上还是按自己习惯，用不同工具（比如上面说到的 Trello、Excel 等）来管理自己的项目信息。

之前公司的平台部也考虑过重新开发一套项目管理系统，但整体评估下来开发成本非常高，所以上述问题就一直遗留至今。

问题 3：项目状态可视化程度差，管理成本高

由于缺乏统一的工具支撑，各部门整体的项目进展可视化自然也是很难实现的。当时如果想收集一份“部门进行中项目列表”，至少需要项目经理 1~2 天的时间才能完成（痛苦的是，对于互联网这种高节奏的开发模式，当收集完项目列表时，列表中的很多项目进展已经有了很大的变化……）。

每天并行开发上线的需求一般有 100~200 个，如果出现紧急需求需要插队，或者进行中项目出现重大需求变更，如何识别相关受影响的项目并进行计划调整，只能靠人工的方式逐个项目收集，对项目经理来说简直就是场灾难。所以，为了降低项目管理/人员沟通的成本，提升团队效率，我们基于 Jira 设计并搭建了我们的项目管理平台（基于 Jira，主要是从成本和速度的角度考虑，因为 Jira 作为商业软件，提供了足够灵活开发的后台配置和 API，可以让我们在只投入少量开发资源的前提下，快速响应我们的需求）。

在解决项目管理问题后，我们延续“打通”思路，把项目管理系统同工程类系统（GIT、Jenkens、持续集成平台等）打通，把信息整合的范围，从项目管理类信息提升为整合“研发全过程信息”。让研发人员可以一站式完成研发全过程的各种操作，而且消除各类信息搬运，基本消除了各类参数复制错误导致的系统故障。

解决思路

第一阶段：项目管理信息打通

改进点 1：减少团队沟通协作成本

特性 A：为各 BU 划分专属项目池。

互联网公司的特点决定了我们大多数项目的模式不是“专门项目，专门的项目团队”，而是“规模小、快速迭代”。所以我们为每个团队，单独划分了一个“Jira 项目”，定义为这个部门“全部研发任务统一管理的项目池”。

- 易于理解，用户在操作项目信息时，只需要在自己所属项目池内操作，不用在多个项目池中切换。

- 部门的结构相对固定，所以初期建立好项目池后，后期的维护工作量很少，有利于减少平台维护成本。
- 由于项目池和业务部门划分有明确的映射关系，所以权限控制、通知提醒规则，以及我们各种二次开发的功能开关，可以基于项目池维度，根据不同业务团队需要，灵活配置各种特性开关。

特性 B：集中管理各类型事务。

我们把需要耗费团队工作量的事情抽象划分成不同的事务类型，包括：

- 业务需求
- 技术优化需求
- 系统线上问题处理
- 日常任务

这样，从宏观的两个维度上，我们就把所有部门（各项目池）的所有类型工作（各类事务）统一纳入了一个系统中集中管理。同之前相比：

- 对用户来说，耗费自己工作量需要记录管理的各类工作可以在一个系统中一站式管理，操作成本和体验上都有了很大改善。
- 各种定义、模板、字段得到了统一，不像之前那样，各系统中记录形式不一致，大家在整理项目报告的时候还要经过“翻译”环境而导致工作量浪费。
- 项目数据归一化之后，后续可以基于一个唯一的数据源生成各种项目报表和视

图，同时数据的归一，将为后续项目度量等工作提供极大便利。

改进点 2：消除管理浪费

之前的项目研发过程中，由于缺乏系统支撑，各岗位人员往往在流程中的各环节，按照自己习惯，使用不同工具管理自己的任务信息，导致了一系列问题：

- 信息搬运的浪费。每个需求都要在多个系统中重复录入需求相关信息（需求描述、负责人、开发计划）。
- 信息不一致。由于在多个系统中录入，经常会出现某个需求在各系统中信息不一致的情况，需要花费额外人力做信息准确性确认。
- 操作入口多，用户体验差。

	需求管理	需求review	开发进展	测试进展	发布
SharePoint	√	√		√	√
Jira			√		√
Mingle			√		
Excel（个人电脑里）			√	√	√
Wiki		√	√		√
SVN	√				
邮件		√	√		
共享目录	√				

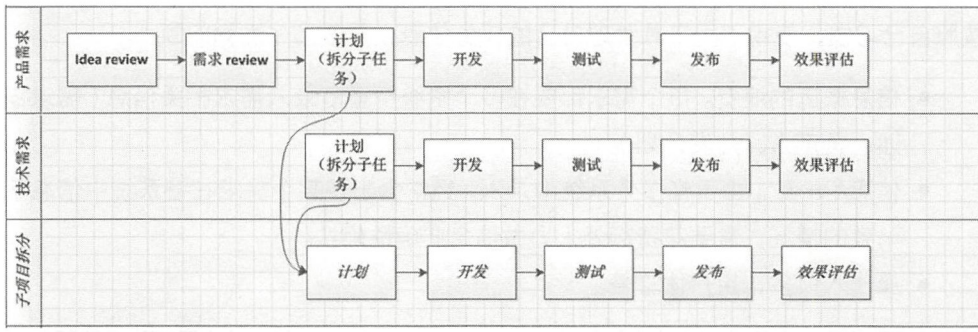
我们的解决思路：利用 Jira 的工作流，实现在一条需求（从 review，到开发、发布）全流程打通管理。即一次业务发布，用一条系统记录跟踪到底。

如下图所示，对一条需求记录，从创建需求开始，依次经过“审查→计划→开发→测试→发布”等状态。流转不同状态时，各环节的项目人员，把项目在这个环节的相关信息录入在这个需求的不同字段中，好处如下。

解决了上面所说的信息重复录入和一致性的问题，确保了不同角色获取的需求信息一致，减少了录入成本。

当大家习惯使用系统并及时更新项目信息后，如果有人想查阅项目的某部分信息，会倾向于到系统中查询，替代之前口头询问的方式。当团队规模比较大的时候，这种“背对背”的沟通方式，要比之前“面对面”的沟通效率高很多。

把公司的标准研发流程以工作流的形式沉淀在系统中，统一了大家的项目运作模式，流程的实施成本大幅降低（不像之前往往要对员工做各种流程培训，以及定期审计流程执行情况）。



改进点 3：易用性

之前接触过很多公司，也都有自己的一套高大上的项目管理系统，但实际上，官方的项目管理工具只是“摆设”，研发团队的任务信息，还是通过 Excel 之类的“本地工具”进行管理。如果去询问研发团队原因，一般都是“复杂”、“不方便”之类的答案（比如，我之前曾就职的某通信行业公司，开发人员经常抱怨的一点就是“改 10 行代码，走 1 天流程”）。

我们认为产生这种现象背后的原因，除工作流、视图等设计与本公司业务情况不匹配外（我们的工作流、字段、界面，基本没有沿用 Jira 的默认配置，都是完全重新自定义的），操作易用性，也是用户是否接纳系统的重要原因。

注：项目管理系统，本身是不直接对业务产生价值的。所以在满足团队项目管理需求的同时，要不断反思用户的使用时长，学习门槛是否还有优化的空间，尽量简化用户的使用，提高工具的“性价比”。

1．尽量简化流转的状态和填写字段。

最早的工作流，从需求提出到发布，一共要走 21 个状态，经过我们反复精简后，只保留了关键环节（需求→需求 review→计划→开发→测试→发布）。

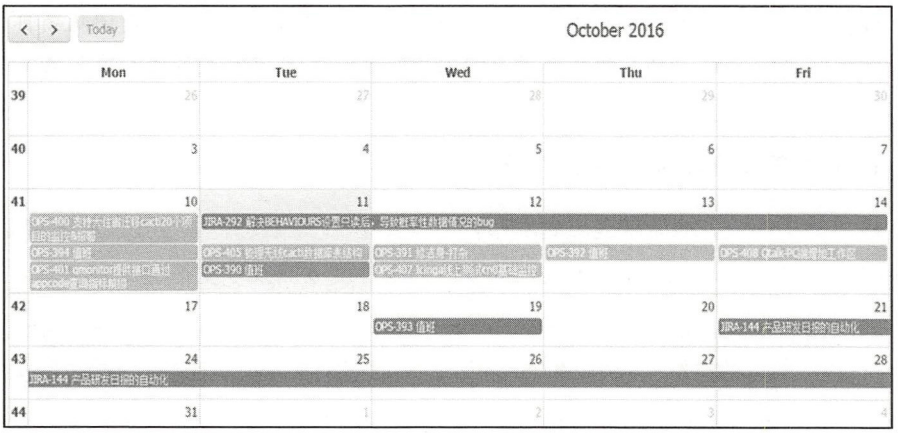
尽量精简字段数量，原则上不要为了管理上的需要，增加各种“XX 产品线”、“XX 业务分类”之类的字段（这种场景，意味着为了管理需要，增加一线研发的填写成本）。在此基础上，还应该考虑如何自动填写项目管理的相关字段，更进一步减少大家在项目管理系统中的操作成本。比如，当我们把项目信息和工程类系统打通后，可以根据代码、分支的变更情况，自动计算并回填项目的“开始开发”和“发布上线”时间。这样不仅减少了用户的使用成本，数据的准确性也得到了提升。

2．减少用户学习成本，提供按用户身份“自适应”的视图。

Jira 作为一个成熟的商业软件，不太友好的一点就是查询操作需要一定的学习成本。所以，我们在首页、看板等视图中，添加了“我的项目”等自适应过滤条件，当用户点击时，可以按照当前登录人的信息，自动过滤当前用户相关的任务记录。

看板视图如下图所示。

日历视图如下图所示。



列表视图如下图所示。

过滤器结果：我负责的需求

类型	关键字	优先级	主题	状态	计划提测日期	计划发布日期	更新日期 ↓
P	DEMO-1167	P4	Jira REST API测试问题，勿删。2016-08-30 00:02:00	测试中	2016-08-10	2016-08-24	2016-08-31
P	DEMO-1152	P4	1212	已排期	2016-08-19		2016-08-19
P	JIRA-144	P1	产品研发日报的自动化	开发中		2016-12-31	2016-08-18
P	DEMO-1124	P4	1212	IDEA 待定			2016-08-10
P	DEMO-1114	P4	test	IDEA 待定	2016-06-15	2016-06-28	2016-08-10
P	DEMO-1125	P4	CLONE - 1212	IDEA 待定			2016-08-10
P	DEMO-1088	P4	test	IDEA 待定			2016-08-10
P	DEMO-1057	P4	1	IDEA 待定	2016-05-07		2016-08-10
P	DEMO-1046	P4	1121111	IDEA 待定	2016-04-30		2016-08-10
P	DEMO-1050	P4	TEST	IDEA 待定	2016-05-04		2016-08-10
P	DEMO-1051	P4	CLONE - TEST	IDEA 待定	2016-05-04		2016-08-10
P	DEMO-1053	P4	CLONE - CLONE - TEST	IDEA 待定	2016-05-19	2016-05-19	2016-08-10

3．简化操作和提高操作效率。

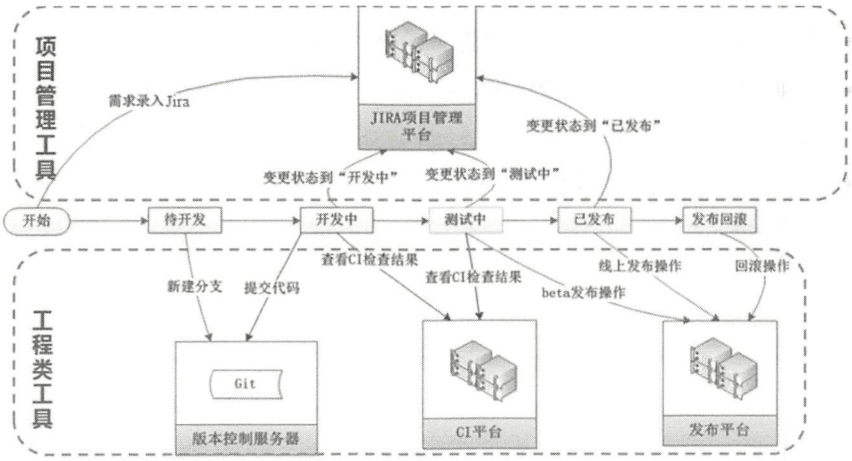
低成本同步物理白板信息到线上系统中：我们很多采用敏捷模式的团队，都习惯用“物理白板+卡片”的方式来管理任务，然后通过每日站会来沟通任务进展，但这会导致大家总是忘记更新系统中的任务状态，影响外部团队获取准确的任务状态信息。在不改变大家工作习惯的前提下，我们增加了一个功能——“对物理白板拍照，然后在 Jira 中上传照片”，就可以根据白板上的变更信息，自动识别并更新系统中对应的任务状态。这样只要每天开完站会后，“值班”同事拍个照片上传到系统中，就完成了本组任务信息的批量更新。

Excel 形式在线批量编辑：另一个初期让人吐槽的地方就是批量修改任务数据时，往往需要一条条输入进去分别编辑、保存，始终不如 Excel 操作方便。后来我们提供了类似 Excel 的在线表格编辑功能（复制粘贴、插入行、筛选、排序……），可以批量编辑后一起提交保存，大大节省了操作时长。

	标识	主题	计划开始日期	计划发布日期	产品经理	DEV	QA
1	DEMO-1689	Jira REST	2017-11-08		系统管理员账号	黎娟jen	
2	DEMO-1688	Jira REST			系统管理员账号		
3	DEMO-1687	Jira REST			系统管理员账号		
4	DEMO-1686	Jira REST			系统管理员账号		
5	DEMO-1685	Jira REST			系统管理员账号		
6	DEMO-1684	Jira REST			系统管理员账号		
7	DEMO-1683	Jira REST			系统管理员账号		
8	DEMO-1682	Jira REST			系统管理员账号		
9	DEMO-1681	Jira REST			系统管理员账号		
10	DEMO-1680	Jira REST			系统管理员账号		
11	DEMO-1679	Jira REST			系统管理员账号		
12	DEMO-1678	Jira REST			系统管理员账号		
13	DEMO-1677	TEST			宋培培		

第二阶段：项目信息同工程信息打通，统一研发全过程的各类操作入口，提高研发效率

我们通过观察工程师的日常开发过程，发现大家在操作各种工程系统（如 Git、jenkins、CI）时，依然存在系统间参数不一致、信息复制粘贴、操作复杂等问题，如下图所示场景。



在项目过程中，工程师需要大致如下的操作：

1．在 Jira 项目管理平台中创建一条需求。

3．Git 拉分支，基于分支提交代码。

3．每次提交代码，各种自动化检查结果（SONAR、UT、自动化测试、code review 等），要到 CI 平台的不同子系统中输入工程和分支名称，检索查看。

4．开发自测的时候，在发布平台要把需求的相关工程、分支名称和发布参数，录入并发布到 dev 环境。

5．自测过程中，如果需要查看自动化检查结果，则重复步骤 3。

6．测试阶段，每次更新版本 QA 的同时、依然要重复步骤 4（发 beta 环境）和步骤 3（查看自动化检查结果）。

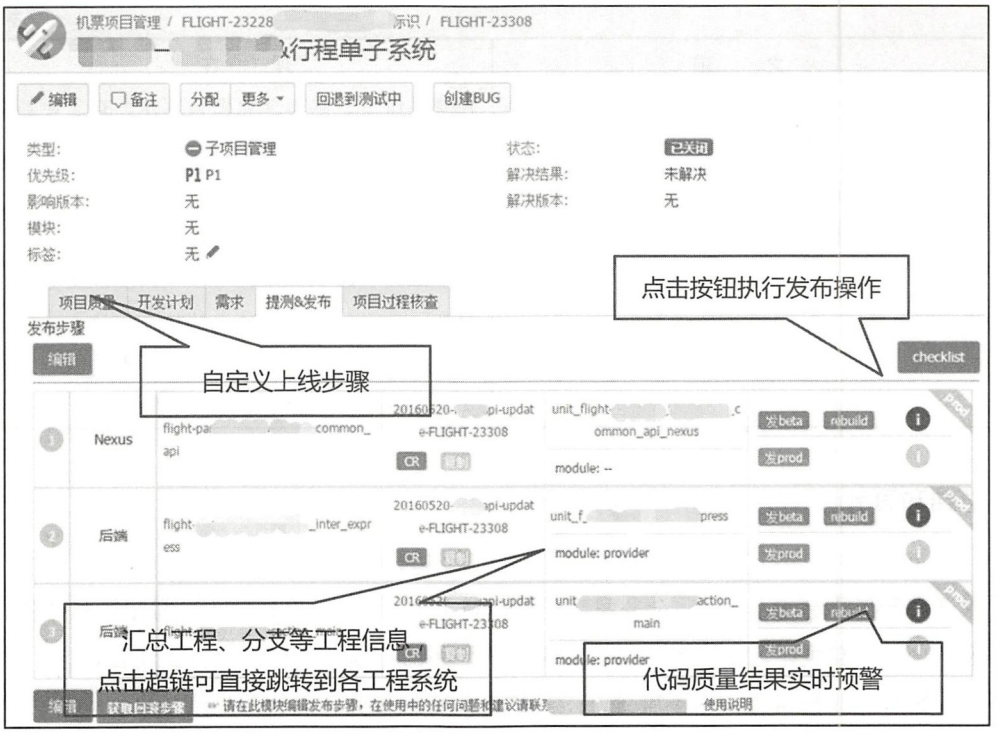
7．在整个过程中，RD、QA 要定时在项目管理平台更新项目状态。

抽象来看，问题本质依然是：

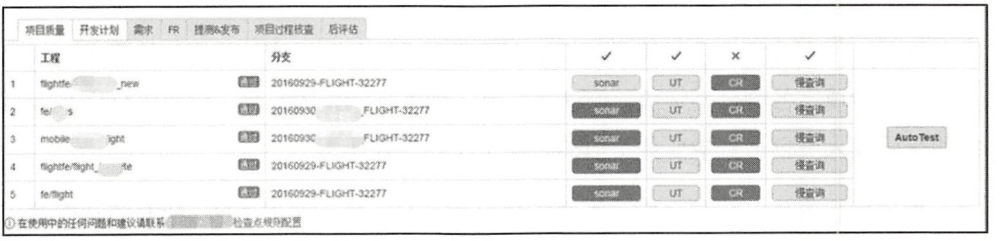
- 信息分散，容易不一致。
- 操作入口多，复制粘贴等手工操作易出错。
- 操作复杂影响效率，当业务压力大时，容易导致流程跳步（比如不按研发规范检查自动化测试结果）。

我们的解决思路如下：

- 在项目信息和工程信息间建立映射关系。
- 基于这种映射关系，当代码变更时，自动触发各类自动化检查。
- 各种工程信息、质量检查结果，按项目维度，在项目信息页面中，统一聚合展示（之前只有项目管理信息）。
- 统一入口，各类常用操作可以在项目管理系统中一站式完成（如发布、回滚），如下图所示。



如果自动化检查出现问题（出现红色叹号），可以一键切换到质量详情页面查看具体 fail 项。



与之前对比，主要有以下好处：

之　前	现　在
各种信息，比如工程名称、分支名称、tag 发布参数等，需要在各系统中重复复制粘贴，经常因为手误，导致发布失败甚至线上故障	信息打通，无需在各系统中重复填写，操作效率高。基本消除了由于操作错误导致的故障

续表

之　前	现　在
代码变更后，需要到各 CI 子系统中人工查询检查结果	按需求维度，各类信息聚合展示，查看更新只需要按 F5 键打开需求页面
RD 要在各系统中来回切换（一般同时开的浏览器窗口数量都是 10 个以上），操作麻烦	入口统一，各类操作一站式完成
	数据间建立映射关系，可以利用工程类变更信息，自动计算更改项目的状态信息，项目相关信息更加客观准确。这些积累的数据，为后续自动化统计度量提供了非常好基础

总结和思考

回想当初，我们刚在酒店 BU 做小范围试用的时候，其他业务部门就过来主动要求迁移到新平台上。半年后全公司都迁移到新平台，无需项目管理人员做审计和跟催，业务部门会自发把团队内部各种事务都录入到系统中统一管理。运行到现在，这个工具已经与大家日常工作密不可分，工具的价值得到了业务团队的高度认可。

总结起来，我们认为这个项目能成功的因素如下。

关于定位：如果把项目管理平台也看成一个产品的话，首先要识别清楚它服务的核心用户是谁。如果管理的诉求和一线研发的诉求有矛盾时，应该如何处理？

比如，很多公司的项目管理部门，为了给老板做各种维度的报表，就要求一线人员在系统中填写各种分类字段，但这其实是一种本末倒置的行为，因为这样做其实牺牲的是研发人员创造直接业务价值的时间（比如写代码）。

所以我们最终认为项目管理系统的核心用户是一线的研发团队，所有的功能都应该思考是否可以把用户在项目管理系统中的操作成本降到最低，做到尽量简单、易用、用完即走。从这个原则去指导所有的功能设计，才使得研发团队欢迎和依赖这套系统。

关于工具的操作成本，还应该遵循一个原则：少做甚至不做低层次的操作（复制、页面切换、查询），让研发人员的时间更多放到设计、写代码等高层次脑力活动中，把人力资源聚焦在创造业务价值的活动上。

工具是否符合公司“本土”特点：项目管理系统中的工作流，实际上是公司日常

工作流程的固化体现，所以不管是用 Jira 还是其他工具，都要结合公司特有的研发流程，进行本土化改造（我们的事务类型定义、工作流、字段都是按特点重新定义的）。只有这样，工具的操作同大家日常工作状态才是逐一匹配的，大家在使用工具时才不会产生“陌生感”。

最后，研发工具不是万能的。想打造高效团队，往往要和组织结构、研发流程、团队氛围等多方面组合提高，才能切实提升团队的交付能力。比如，我们在实施项目管理平台的过程中，还要结合不同团队情况，开展组织结构调整、业务系统解耦重构、产品需求拆分等其他改进实践。所以，遇到问题，想都通过工具解决，或是都通过流程规定解决的狭隘思路都是不可取的，应该从多个改进维度中综合考虑最适合的解决方式。

NO.4
产品篇

前文介绍了我们做“大”产品的成长历程。那么，如何善用敏捷开发快速迭代和验证产品的价值？如何应对自上而下、打配合战、自下而上的不同类型的产品？如何主导推动大型产品运作？如何创新驱动产品迭代升级？在本篇中，或许可以给产品经理们一些启发。

第 6 章

用敏捷思维做大产品

善用敏捷开发构建、快速迭代和验证产品的价值，推动小产品做大，并不断创新，驱动产品迭代升级。

6.1 小产品如何做大

从产品发起的三个维度，部门领导安排自上而下实施，部门间互相配合共同打造，员工提出自下而上争取。下面分别就我们做过的产品现身说法，讲述我们的推进历程、碰到了哪些问题、我们又是如何应对的。

6.1.1 自上而下的产品，如何快速实施做大

有些时候，老板会突然提出一个产品构想，在大公司里，这种情况很常见。老板今天提这个想法，明天说不定有什么灵感又提一个想法，如果产品经理全盘接肯定扛不住，但是不接的话就是不服从。某些时候，老板又寄予了很大的希望，希望能把一个产品构想做起来，做大做强，所以给产品和技术团队很大压力。如何快速贯彻老板的想法，尤其是在一堆高优先级的产品需求排队的情况下，如何将想法快速上线，能及时看到效果并迅速给老板反馈，以便老板能加大资源投入或者是彻底断了念想，这对团队是个极大的挑战。

微领队这个产品就是大领导提出的，部门自上而下推进，快速上线，不断迭代，由小做大的一款典型产品。2015 年，正是微信群火热的时候，大家几乎都热衷于微信群聊，简洁的聊天窗口以及简便的操作方式，让这款聊天产品迅速火爆。此时，一位高管提出，要为下了携程订单的行中用户建群，为他们提供服务和支持，这就是微领队点子的雏形。

领导的这个想法很有理论支持，因为创业公司没有流量，但是携程有流量，每天上千万的日活跃访问用户量，每分钟都有用户下单，如此庞大的行中用户规模提供了基础的保障。于是，这个任务安排下来了，当时的自由行事业部服务部门承担了最初的微领队试运行工作。

部门的资源调配都是看产品及项目经理 ROI 的，这个产品最初的阶段优先级根本排不上，基本所有的项目参与人员，都是兼职的。运营 1 人，服务 1 人，其他业务和产品人员兼职帮忙，资源十分有限。但是在这种情况下，还是按照敏捷的方式，每周做几个小需求，保持了每周一次的会议，讨论现状和可创新的点。

开始挑选了几个热门的目的地，试运行 1~2 个月后，感觉效果还可以，老板认为有利可图，可以继续做，于是给加资源，招了个 3 级员工，带着另一个部门转岗的 20 人开始做这个产品。服务的目的地也从几个扩展到几十个目的地。

当时，微领队对技术产品的依赖并不是很强大，因此产品经理从敏捷角度帮忙做了一些后台的小功能，优化短信邀请、群管理等，并开始研究可用于微信的小外挂等。后来这款产品搞得如火如荼，一来发现很受用户欢迎，极大地解决了他们的痛点，二来是自己干得也很开心，大家干劲十足。

2016 年老板安排了一位运营总监全职负责微领队的项目。从那时起产品、研发、运营人员齐全，微领队有了自己的全职团队。公司也安排了一位产品总监负责微领队的产品工作，实线汇报给职能领导，虚线汇报给项目负责人。虽然正式的产品经理只有 2 人，一个负责前台，一个负责后台，就是这样极为精简的产品经理配比，持续了一年半的时间，使业务量扩展了 *N* 倍。我们和研发总监带领的研发团队，合作十分紧密，一直按照敏捷模式，两周左右时间一个迭代，快速验证。

微领队的业务团队是一个敢打敢拼有想法的团队，产品经理一直和业务与服务团队在一起，持续头脑风暴，持续创新，不断试错，不断校验自己的很多想法。在最早的基建阶段（APP 中 IM 的体验还比较差），我们需要有一些行中功能帮助用户留在微

领队中，附近旅友、行中的结伴等功能差不多是那个时期的产物。等到微信和 APP 中 IM 的成熟，微信和携程 APP 中服务的用户比例基本趋于稳定。

产品经理的设计方向就变成了提升行中的服务体验，并尝试一些轻社交属性。旅游圈、SOS、清单等功能是这个阶段的产物。其实除了大家看得到的一些功能，还有很多是非团队成员不曾见到过的。

我们的点子基本是一个漏斗模型，有八成的创意，基本都没有落实到产品需求说明书中，就在讨论中被标记了很低的优先级或很低的可行性。讨论出来，觉得可行且价值认可度高的，会进入产品经理的 Backlog 排序，并进入产品设计流程。

我们严格遵循 MVP 的原则，基本上每个版本中推出的新功能，会保证流程是通畅的，然后用一个版本来看用户是不是买单（具体来说就是看该模块的点击率和活跃度，如果是售卖的，还会看订单转化率等）。如果基本符合期望，就追加资源，进行产品优化，或者继续进行 AB 测试，持续改进。如果没有达到预期，基本就会直接下掉，或者停止研发成本的投入。就这样，微领队产品越做越大，2016 年获得了携程集团公司级项目奖励，那一年我们的目标是持续扩量。

产品在这个阶段进行了很多的流量合作，增加了更多的外部入口，并且把服务对象从正在旅行中的用户，扩展到部分的已经下了订单，还在出行前的用户。

2017 年，除行中的服务外，还有行中的售卖。突然间，2017 年重新确定目标时，目标变了，之前是看 GMV，所以大力投入资源，2017 年的目标是看 ROI，小伙伴们应该都了解，GMV 和 ROI 在某种程度上存在着冲突，测算了 2016 年的 ROI，仅为 0.3，2017 年的目标提升到 1，包括服务团队、业务、运营和研发、产品成本，同时 NPS 数值提升到 35 以上。

这对我们团队是个很大的挑战，GMV 不能跌，ROI 要翻 3 倍多，意味着成本要控制，产出要提高。对于一款大产品来说，目标调整，相应的组织架构也要调整，产品方向也要调整，运营人员要承担翻倍的指标。产品和运营人员重新确定方案，通过优化后台产品，提升人员效率，同时升级运营模式，与目的地合作商共同管理微信群，经过半年的努力，目前微领队产生的 GMV 达到了 2016 年同期的 3 倍，ROI 已达到 1.05。

在达到这一阶段后，产品的职能就与之前有了一些变化，我们也开始测量模块的

收益，如纯财务维度的。附近热卖就是代表的功能，包括在天气预报中植入保险售卖。在这个阶段，我们进行了一轮非常谨慎的测试，测试我们一些完全与财务无关的宫格对售卖的影响。在商业智能数据组的支持下，证明了活跃度和订单的相关性。PM 的 Backlog 中，对于售卖提升和效率提升的相关需求会被提升到更高优先级。

除了可见的客户端产品，我们在服务运营的后台投入也非常多。从开始纯人工模式的建群、加好友、发内容，到目前绝大多数的筛选数据、建群、发邀请、加好友、自动推送等，都已经是系统自动的了。

2017 年，一位技术总监加入，这位技术总监很擅长机器学习，他带领的团队在自然语言处理、个性化推荐方面很有经验。技术和产品人员很快将 AI 应用于微信群监督和智能产品推荐中，原先需要通过人工抽查和群内监控方式的工作便不再需要了。通过产品方案不断提升服务与运营的效率，解放出来的服务生产力，又可以用于更多服务的创新。

微领队的成长几乎是一路的创意与验证，一路实践着精益创业精神与敏捷研发模式，在没有可抄袭模式的情况下，一路跌跌撞撞走到今天。路途中必然有过坎坷曲折，有过挫折和迂回，然而伴随它成长至今，留给我们的都是满满的收获与感恩。能与这样一个充满未知挑战的产品同行，十分万幸。更要感谢的是老板一直以来的支持和信任，还有团队的小伙伴们的辛苦拼搏，忘不了半夜三更群里讨论问题的火热场景，这种热情一直激励着我们，坚持努力把产品做得更好。

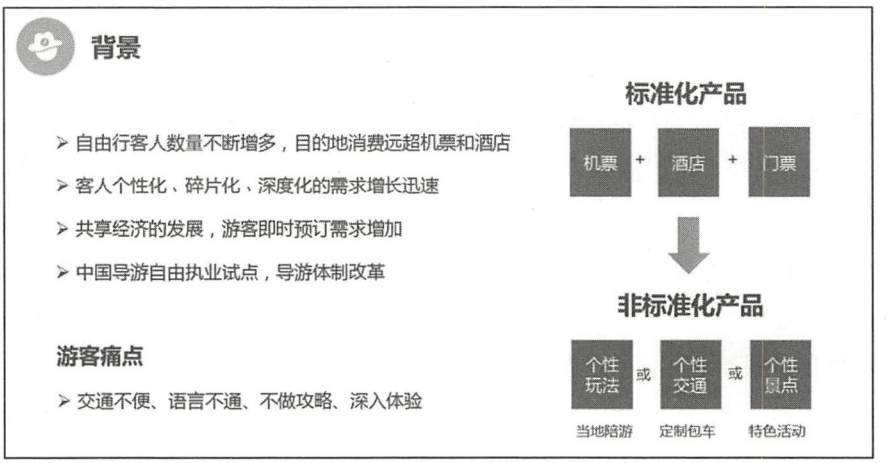
6.1.2　打配合战的产品，如何快速上线做大

一些产品的诞生过程很机缘巧合，一个集团公司下面几个事业部恰巧都有想法去做，但是自己只拥有其中一块资源，这种情况下，大家意愿相同，目标一致，双方都有动力去推进，剩下的就是如何协调，互相配合跟上节奏，共同完成。从开始一个部门进行，到后续和其他部门合作，共同推动完成一款优秀的产品。

2016 年，国家旅游局开始大力宣传和推进“导游自由执业”。年中的时候，一位高级副总裁把这个作为一个研究项目交给了度假事业部 CEO，该 CEO 就把 C2C 服务的项目放在我们部门进行探索研究。

业务这块由一个总监来负责，他有服务和业务运营等多岗位的经验，是一个非常

有激情的人。在还没有技术研发配给的时候，业务先行，商拓、公关、基础调研都是他一个人做，事事在第一线。



目前携程现有主要业务，如机票、酒店、门票都是标准化的，但是用户到了目的地后，如何从机场到酒店，如何到景点，在景点去哪里吃饭，这些都是问题。刚好国家出了导游自由执业的政策，正好现在流行共享经济，可以考虑做导游平台。

项目正式立项后，我们有一个管理培训生出身的产品经理开始跟进。从 6~9 月，这个项目就依靠着一个业务人员和一个产品经理的配置开始运作。一点不夸张地说，他们几乎天天加班。

那时，在另一位同事的建议下，人的系统搭建在度假社会化平台，商户端直接使用独家 VBK，货架进一个线上系统，订单进另一个线上系统，这样可以避免重复建设。同时，微领队需要一个货架，我们的平台可以兼容微领队的需求。

事物都有两面性，由于涉及的系统太多，产品链条太长，9 月第一期上线的基本是一个“待纠错”的版本，也没有太多订单，但我们的业务就同时开始运行起来了。桂林、上海等多地的向导注册在持续增加，总体来说，量增长太慢，如果向导不能覆盖到国内的大部分目的地，有意愿尝试找当地向导的用户也会因为找不到而不再来。

其实在我们上线之前，是有向导资源的，携程攻略社区很早就开始做当地向导业务，但仅仅是作为一个向导的展示平台，用户浏览自己的旅游目的地，找到向导资源，然后线下让向导提供服务，在攻略中以提供信息为主。携程如此巨大的流量，却没有实现流量的变现，攻略的产品团队也意识到这个问题，想要建立订单流程，但是零基

础重新搭建一套订单系统成本太大。

我们上线后，就发现和攻略社区的思路是不谋而合的。在这位高级副总裁和攻略社区 CEO 的撮合下，大家达成了一致的目标，来谋求公司利益的实现。

目标一致是个好的开头，后续还有很多需要两个部门共同配合的事情。打配合战，就要求我们和攻略两个部门步调一致，两个开发团队，几拨业务线，产品、技术、运营、市场等人员都要紧密契合。为此，两个部门的主要负责人开会沟通，确定了沟通模式及作战打法。每周两个部门的 PO、开发负责人、测试负责人、运营负责人一起开会，讨论每个部门当前工作的进展及进展中碰到的问题。然后每个部门自己也开自己的固定会议，整体确保信息对称，沟通渠道畅通。

我们和攻略两个部门在需求优先级确认时，先确认好两个部门之间的需求依赖关系，确保有依赖关系的需求在同一个 Sprint 内完成，同一时间节点发布。经过一轮轮的讨论和双方运营人员的协调，最终通过几个月的时间，我们将攻略的向导平稳过渡，重新通过商家端注册为携程当地向导。旅游首页和攻略社区目的地首页如下图所示。

至此，携程“当地向导”不论是业务模式，还是产品，才算基本确定。可以对携

程各个来源渠道的向导资源进行规范的认证和管理，规范了上货，统一了预定流程、订单流程、结算制度等。

2017 年 1 月，我们从 0 佣金，变为开始收佣金。这个春节也是我们产品基建搭好之后的第一个旺季，当月营收基本已经能养活业务运营团队，春节的峰值为数千单，系统未出现重大问题。

很快，我们 2017 年年度业务指标目测可达成，于是目标就被提升为 GMV 1 亿元。从那时我们开始进入加速轨道，春节后，给产品团队配备了一个经验丰富的高级产品经理，又增加了一个产品经理，业务团队扩张到十多人。

后来的三四个版本里，我们逐渐使用多团队配合，产品加快了迭代速度。把依附于巨人系统上的向导平台的差异化需求逐步实现，更好地支持业务发展。

产品架构上，首先完成了价格套系改造。在一个版本内完成了价格套系改造，同时 20 天内完成了“旅途聚会”品类的上线，涉及 7 个产品组，8 个研发组。在项目经理和产品负责人的协调下，这样的工程量以极高的质量如期交付了。也是这样一个周期后，技术和产品的团队合作力进一步提升。

最早我们的向导首页用社会化人的首页，那个首页最早是为旅行顾问设计的，而向导首页承载的更多是商品，类似店铺页面。因此，产品重新设计了向导主页，并在后期与微领队、直播团队合作，加入向导的旅友圈和视频等功能，使得个人主页更立体化，更具有亲近感，提升了转化率。

订单层面，在供应商参与的情况下，议价和改价都是常见的。产品进行了“先下单后支付”的下单与支付流程改造，并且支持下单后改价，大幅提升了向导与客户的交易效率。

随着向导规模和产品数量的进一步增加，对于产品审核的效率要求也逐渐提上日程。产品审核框架从无到有，通过审核结构化与自动化，从日均审核量几百条/天，增长到上千条/天。一季度时立下的 1 支亿元的目标，我们只用了半年就已达成。产品和业务于是又把自己的 OKR 和财务指标进行了重新设定。产品历程如下图所示。

这款产品能有今天的成功要感谢兄弟部门的支持和合作，跨部门产品的合作需要统一目标，找到大家共同的利益点，这样大家都有推动的积极性和把事情做好的动力。确定目标后，接下来是分工和主导，尤其感谢攻略社区的同事，主节奏是我们度假事业部来把控的，各个相关事项攻略社区都积极配合。这样在进行中就减少了很多扯皮的事情，我们不但打造了一款成功的产品，更重要的是我们在合作中互相组成了一个幸福融洽的团队，结交了一群朋友。

6.1.3　自下而上的产品，需求的提出与推动

在 IT 公司干了几年的人士应该都了解，大老板才是幕后的需求发起方，基本上，所有的产品经理都是跟着大老板的指挥走，去进一步分析老板的构想，并进行需求的设计与分析。

有时候，产品经理自己有了一个很好的想法，想试一试，但是一时又不好评估能带来多大的收益，这种情况下如何获得老板的支持呢？如何拿到资源来尝试呢？当然，这种情况一般在大公司相对容易实现，小公司里的工程师都是身兼数职的，老板不敢划拨资源来冒这个风险。在大公司因为分工比较明确，有的事情让每个环节上的工程师抽出点时间来做倒也未尝不可。

我所在的团队负责的产品——旅行日程，其诞生历史也可谓曲折。从当初一个小想法，到现在占据了携程 APP 首页的一个关键宫格，整个团队奋斗了 3 年多，实属不易，最关键的是要感谢当初创意的提出者，以及团队的产品负责人，没有负责人的节奏把控，这个产品根本成长不起来或者长到一半就夭折了。

几年前，大概是 2014 年年初，携程无线事业部的一个很年轻的产品经理向领导提了个建议，说可否做一个日程类的产品，可以把用户行中所有的相关信息串在一起，

这样用户打开携程 APP 就能看到相关的机票、酒店等的时间、地址、电话等信息，用起来会非常方便。

这个同事当时看到一个网站——Trip it，主要是做旅行计划的，可以输入并编辑自己的旅行信息。这个同事自己用了，觉得不错，然后就在产品会上提建议，当场就被老板给拒绝了，理由很简单，没资源，优先级不高。当时携程的重点是从 OTA 往 MTA 转型，几乎所有资源都投入到了 APP 的设计与建设中，确实没有余力。

这个产品经理很坚持，想想只靠自己一个人搞不定，于是搞了个用户研究调查，发放了几百份调查问卷，同时访谈了一些资深客户，拿到一手资料后，做了一份数据分析报告，在产品会上又一次提出这个建议，同时拿出数据分析报告，在一番合情合理，言之有物的分享后，征得了部门 CEO 的同意，但是，因为需要在携程 APP 首页开入口，还需要在公司级的产品会议上评审，征得大老板的同意才行。

尽管做了很充分的准备，精心制作了 PPT，但是在公司级的产品会议上一下就被大老板问住了，这款产品的终极目标是什么？分几个阶段？阶段性目标是什么？每个阶段需要投入多少资源？收益有多少？大老板讲究的是投资收益比，一切都要求量化，这个创意还没有精细规划到这一步，自然没法过大老板这一关。

好在产品经理坚持，部门 CEO 也很支持，划拨了两个产品助理，先理了一下产品的 Road Map，明确目标等。过了一周，基本的产品框架成型了，在部门 CEO 的安排下，又一次在公司级的产品会上评审了。

目标很明确，传统的旅游订单管理关注订单本身，订单是 OTA 企业和消费者之间契约关系的凭证，所以在网站、APP 等订单界面中，往往只是依次罗列订单信息，并且偏向于强调订单金额、支付方式、配送方式等这些在预订前过程中的信息确认，有信息免责的意味。而从一站式服务平台的责任出发，携程并不认为预订的完成即是服务的终止，对用户的服务更应该延伸到预订之后的出行帮助。从这个角度出发，订单中的出发时间、地点等行程信息，是用户在实际出行过程中会更关注的有用信息，而“旅行日程”便是秉承这个使命诞生的，其整合了订单中的行程信息，为用户提供了统一的行程管理。

这次评审相当成功，再次说明功夫不负有心人，大老板同意了，但是只给 3 个大版本的尝试时间。当时携程 APP 整体走敏捷开发，半个月发布一次大版本，也就是说，要在一个半月后看到效果，否则就下掉。

时间很紧迫，一定要小步快跑。一方面，由于入口本身以及业务性质决定了用户并没有耐心去浏览或操作一些主场景以外的信息。因此，设计目标一定是单任务单场景，这也符合便携设备的使用习惯。另一方面，一定要做最容易看到效果的东西，确保这款产品能活下去。我们采用了卡片设计作为整个旅行日程设计的基础，卡片设计的优点是它有着天然的局限性，正是这种局限性使得其他部分的设计变得简单许多，不论是信息还是排版，都会强迫我们做一些取舍，让整个任务场景变得更简单。

纵观携程的业务，公司从旅行社起家，酒店和机票是两大个柱产业，先做这两个卡片。同时部门内部调配了开发人员、测试人员、设计师，组成一个敏捷团队。产品、设计、研发、测试人员共同汇报给一个负责人，实线汇报给负责人，虚线汇报给相应的职能领导，Scrum Master 的角色由部门的项目经理兼任。

最初的构想是将机票、酒店订单的相关信息写在卡片上，确保先让用户能看到，这就需要跨部门的沟通。找到了机票和酒店部门相关的产品负责人，说明了相关情况，对方态度还不错，但是一谈到需要接口支持、增加参数等，回答就是资源紧张，排期排不上。

相信这种情况各位工作多年的人士很熟悉，大家都忙自己部门内高价值、高优先级的需求，大家都背着 KPI，自己的事情都忙不过来，哪里有空来支持别的部门？修改、联调、测试等，整体算下来要投入的资源不少。

我们回来拉了自己部门的几个产品经理一起聊了一下，要想让对方部门支持配合，得找到共同的目标，找到共同的利益，这样对方才有动力配合。想了很久，那就交叉推荐吧，对方部门也愁流量和订单，我们来推他们的产品，双方互惠互利。这一招果然很有效，我们和机票、酒店部门的产品经理很快达成一致，他们配合我们提供相应的接口，我们推荐他们的产品。

双方约定了联调时间，每周开会沟通进度及问题。很快，第一个版本发布了，虽然是 AB 测试，流量只切了 20%，观察了几天数据，数据很好看，比预期的要好很多。同时，带给机票和酒店的引流也不低，于是流量全切。第二个版本，产品样式进行了优化，然后发布上线。第三个版本，着重进行了技术改造，优化接口性能。

三个版本的时间到了，第三个版本发布后一周，根据一周的数据情况做了细致的数据分析，在公司级产品会上进行了专项汇报。阶段性 KPI 非常好，大老板也相当满意，然后增加了资源，开发团队扩张到 7，8 个人，并有 3 个测试人员，真的是可谓

兵强马壮。

尝到了起初的甜头，我们更加喜欢敏捷开发了，一定要小步快跑，一定要把大产品切割成独立的小产品。后续的版本又陆续增加了火车票卡片、用车卡片、度假卡片，可以支持导入日历，用户可以自定义卡片。

纵观旅行日程这款产品的成长历史，如何把一款产品做大做好？一定要坚持小步快跑，始终坚持围绕用户的出行场景；拓展预订后的信息及服务，做好场景分析；数据要及时迅速准确的反馈，要在每一个阶段证明价值；快速迭代，验证。

经过了近两年的发展，随着旅行日程的使用量不断提升，新的业务不断涌入，局限性的缺点也暴露得异常明显。单任务、单操作、信息简单，整个框架已经到了极限，根本不可能再加入更多信息卡片展示，整个设计团队也有了一些迷茫，认为不应该为了业务导致整个设计变得臃肿、零散。

设计团队负责人经常会提醒整个团队，设计即是克制，即业务和用户场景的平衡。当小产品做大做强后，再进一步进行主干上的优化就比较困难了，这时候创新求变才是我们的价值。我们一直在寻找创新点，我们也在坚持创新，旅行日程这款产品我们一定会越做越好。

6.2 大产品管理之道

主导推动一款大产品的实施，是十分不易的。为此，我们提出了产品价值演化模型，用以使团队目标达成一致，辅以委员会的“履带式”的牵引和保护，并成立跨公司组织的项目团队，有效地保障了产品在组织层面和集团跨公司的顺利推行。

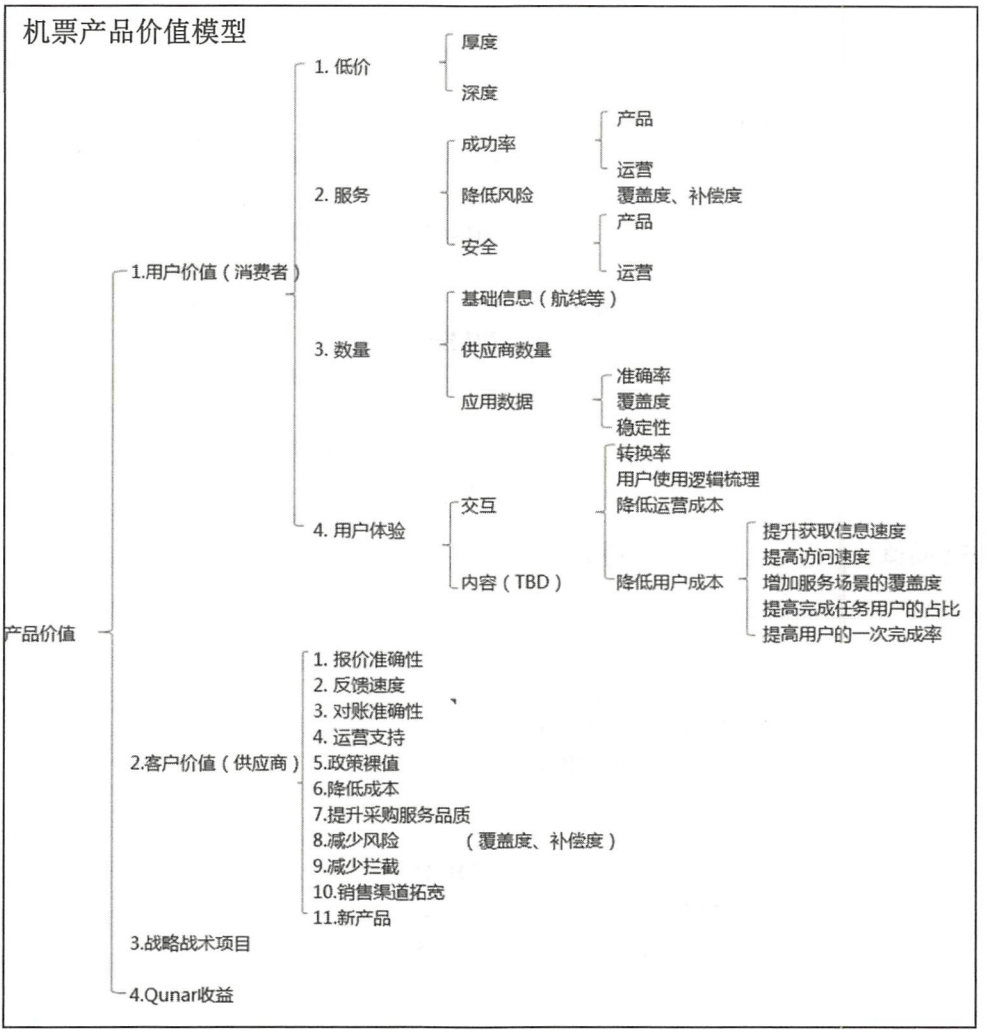
6.2.1 产品价值模型演化和实例

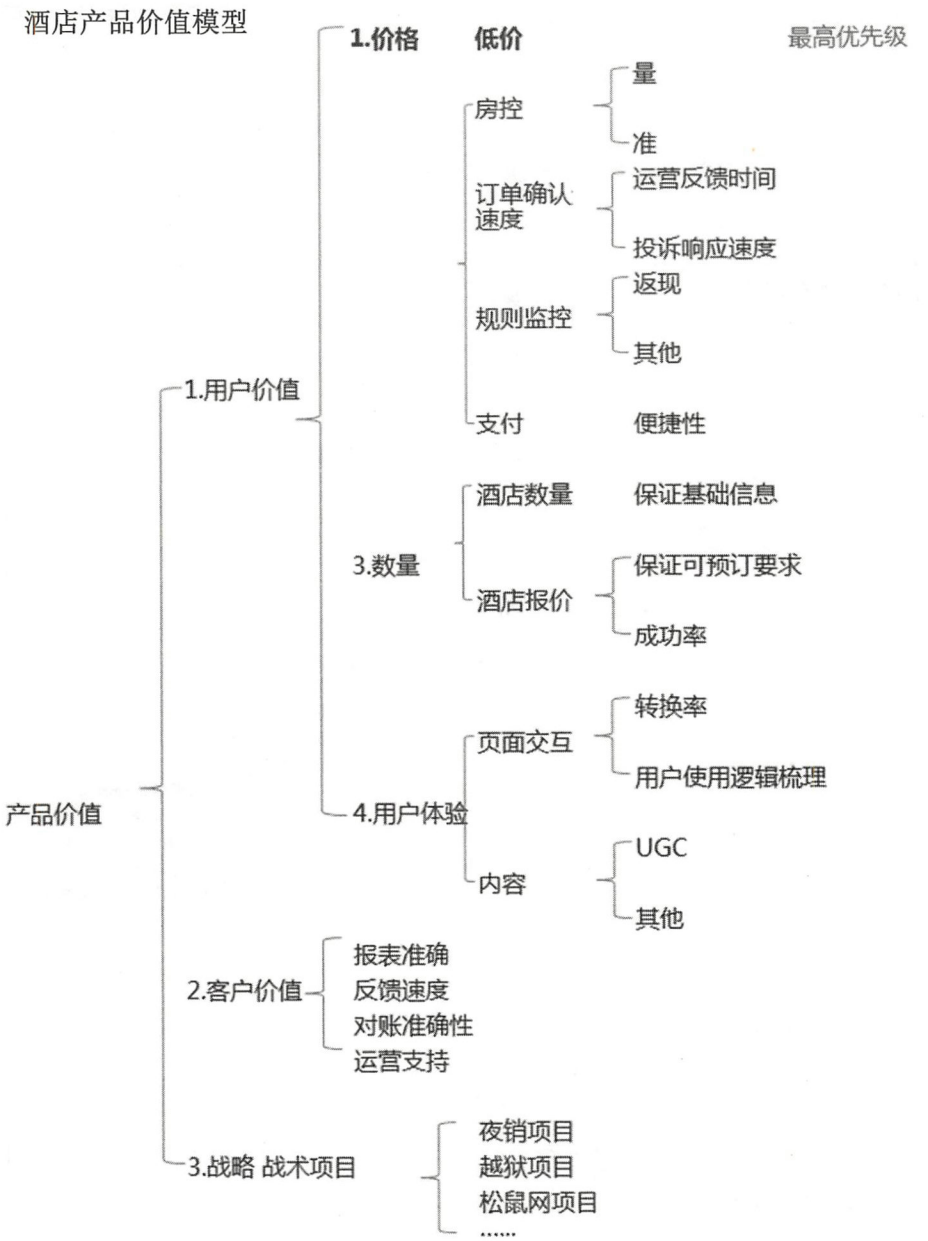
每个公司员工都知道，消费者第一、客户第二、公司第三是公司的核心价值观。我们的产品价值模型的建立过程也深受其启发，并且通过价值模型的持续迭代，帮助团队和员工更加深刻地理解公司文化，二者互为依存。

最初的产品价值模型的雏形，由 CEO、产品 VP、各产品线总监及各团队核心产

品经理根据当时的战略发展方向，共同讨论确定，并以此作为业务运作的依据。之后经历了几个阶段性的演化，才逐渐成熟。

产品价值模型从用户价值（消费者价值）、客户价值（供应商价值）、战略战术项目 3 个大的方向为出发点，逐步具体和演化。下面展示下各产品线的产品价值模型，机票和酒店的如下图所示。





交易类产品的演进过程

首先，产品萌芽期：快速建立用户的核心价值。此阶段集中精力在价值模型中的

用户价值（低价和覆盖），致力于给不同类型的用户提供性价比最优的产品。通过这两个方向快速建立起产品的核心竞争力。

其次，产品的快速发展期：满足用户核心价值的同时，将重心逐步向价值模型中的客户价值以及用户价值（用户体验和服务）转移，从用户及客户角度出发，提供最流畅的体验和优质服务，通过用户体验和服务留存更多的用户。

最后，我们在潜行过程中不停地去尝试和学习各种创新思维，并且应用于产品端，籍此诞生了产品价值模型中的动力——战略和战术项目。此类项目可以冲破内部各种束缚，短期内快速得到验证和成长，从而逐步成为业务和产品增长的新动力和新突破。

产品的整个演进过程随着公司战略的调整有条不紊的进行，同时在各业务开展过程中，不断尝试、容错、突破循环，将业务推上一个又一个台阶。逐渐成熟的产品价值模型，它明确了业务在不同时段的突破点和重心，激励团队形成合力，攻克难关。又在其不断的演进过程中，发挥对团队产品和业务发展的指导作用。

产品价值模型对于团队最大的作用是什么？

1．目标一致

2．信息透明

3．参与决策

以此做到把有限的资源放在最有价值的产品项目上，期望实现价值最大化。

如何做到呢？通过产品价值模型确定产品线当下业务发展的重心和突破的方向，由此制订出具体的业务目标（产品、运营和市场等），再进行下一步分解，即推进和执行每一个重点的产品项目时，就需要把有限的资源放到这些项目上。简而言之，这些项目会占据最主要和大多数的技术资源，同时通过产品优先级队列和排期进行实际的体现。PMO 会有几种数据监测，实时反馈给产品总监，以供调整、矫正和决策依据，真正做到信息团队所有成员之间的透明化。同时通过公司大声说话的文化，让团队所有成员参与到产品业务方向的策略、目标制订和调整过程中。最后在评估环节来验证当初的目标达成状况，总结经验教训，触发新的创意的诞生，从而实现产品生命周期的闭环。

那么，在价值体系的指导下，我们都做了什么？

部门在创建之初，核心成员一致认为，应重点先做产品低价和数据覆盖，起初我们通过人工定期抽测监控国内和国际机票价格的准确率、可预订率与低价竞争力，以及国际航线航班完整性等，有针对性地逐步扩大我们低价产品的深度和厚度。并在2012年建立自动化监控系统，全方位地监测竞品的价格优势，同时定期汇报网站的访问频率、接口响应速度、容量等健康状况。为了及时获取重要信息，我们做到了人工监控和系统监控双管齐下。

2012 年，机票逐步将重心转移到了用户体验，再到用户服务，期间成立了用户产品部，投入了大量产品技术资源，专做用户体验相关项目，经过一次又一次的改版和优化，转化率大幅提高。机票团队对于用户服务的投入也巨大，如我们的用户服务中心、服务平台、消费者保障中心、代运营等项目应运而生。在机票的业务发展道路中，战略战术项目也发挥着极其重要的作用，如飞悦、约票、资金和信息安全、机票包装、包机等项目，虽然有的成功，有的失败，但都在丰富产品类型、提升用户体验等方面做出了巨大贡献。

机票 2015 年产品价值模型实际项目分布，如下图所示。

产品线	低价	服务	数量	用户体验	性价比	客户价值	战略战术	Qunar收入	
国内机票	87	110	17	146	1	193	56	5	
国际机票	80	115	44	161	65	107	14	2	
航空公司	0	0	0	0	0	0	0	0	
交通金融	21	16	12	11	0	38	6	3	
供应链管理	29	26	5	30	7	60	22	0	
服务平台	0		1	46	0	63	5	0	
合计	217	267	79	394	73	461	103	10	1604

同样的，酒店部门也经历了几个阶段的演进。最初的业务核心由覆盖转移到了低价，核心的转移不代表放弃其他用户的价值，如用户体验和服务等，只是我们将更多的资源和精力集中攻克现有的难题，如酒店当时的 PS、返现、红包等都体现了低价策略。当然我们也会阶段性地将重心转移，如酒店系统的重构、实现直销 TTS、OTATTS 等到 QTA 的转移，以及为了提供更好的服务做的 EB 等项目。

在提高公司整体用户产品服务方面，由项目管理部牵头做了退款（覆盖酒店、机票、团购、度假、门票和火车票等）、出票（机票）、返现（酒店）和到店无房（酒店）等节点进度展示项目，同时建立了面向用户的用户平台，借此推动各业务线，解决用户服务方面的产品痛点，持续关注服务质量和效率的提升，这都是整个公司在用户价值和服务的投入。

酒店搜索和交易系统的具体实例（2012 年初），如下图所示。

产品线	产品目标	产品现状	产品价值细型	项目数量	重点项目
酒店搜索	最低价低于ctrip价格80%的酒店占全部ctrip有报价酒店的比例达到80%	目前比例37%	用户价值-低价		1
酒店搜索	整体可预订率99%	目前报价可预订率95%左右	用户价值-数量-酒店报价		
酒店搜索	有报价酒店数量：非bnb类酒店6万家可订，bnb类民宿5万家可订	目前45000家左右，3000家左右	用户价值-数量-酒店数量		
酒店搜索	酒店搜索D2B0.25，酒店搜索首页搜索框S2D2.5	目前D2B0.18，首页搜索框S2D1.46	用户价值-用户体验-页面交互-转化率		
酒店搜索	酒店基础信息99%准确和覆盖	准确率是 96.20%，召回率是93.59%	用户价值-数量-酒店数量-基础信息		2
酒店交易系统	有价代理商数量	90～100	用户价值-数量-酒店数量		
酒店交易系统	有效报价酒店数量（不去重/去重）	11000～13000/3500～4000	用户价值-数量-酒店报价		
酒店交易系统	全站最低价酒店数量	900～1000	用户价值-低价		
酒店交易系统	全站较携程低价竞争力	?	用户价值-低价		
酒店交易系统	Booking到订单转化率	15%	用户价值-页面交互-转化率		1
酒店交易系统	有效订单占比	45%	用户价值-页面交互-转化率		
酒店交易系统	支付成功率	47%	用户价值-服务-支付		1

注：以上所有项目的最终达标率为 78%，基本达到预期。

酒店-用户价值-服务项目的具体实例（2015 年中），如下图所示。

产品价值	业务目标	项目	项目状态	产品经理
用户价值-服务	用户服务579战役	APP投诉渠道优化	idea中	滕晓阁
用户价值-服务	用户服务579战役	未按规则到店订单详情优化	需求中完成	刘莹
用户价值-服务	用户服务579战役	今日入住订单待真理疆	开发中	杨依柳
用户价值-服务	用户服务579战役	15分钟投诉订单转催优化	开发中	张驼
用户价值-服务	用户服务579战役	APP到店时间优化	开发中	滕晓阁
用户价值-服务	拒单追回项目（拒单用户在Qunar下单率达到66%）	赔付红包	需求完成	建洋、徐娟、崔子豪、欧阳琴
用户价值-服务	拒单追回项目（拒单用户在Qunar下单率达到66%）	追加支付	需求完成	张一飞、王佳宇
用户价值-服务	拒单追回项目（拒单用户在Qunar下单率达到66%）	赔付流程	需求完成	闫丽霞
用户价值-服务	拒单追回项目（拒单用户在Qunar下单率达到66%）	拒单A转B	需求中	徐娟
用户价值-服务	运营效率提升 50%	新单组向导式操作流程	idea中	订单组
用户价值-服务	运营效率提升 50%	拒单组向导式操作流程	idea中	订单组
用户价值-服务	服务提升 降30%电话订单比	新增红包返现问题标签及返现流程touch页	需求中	王青、薛雨园、卢颖杰、刘彦玮
用户价值-服务	服务提升 降30%电话订单比	现有标签及FAQ优化	需求中	欧阳逸舒、李钦
用户价值-服务	服务提升 降30%电话订单比	自助变更取消优化一期	需求中	卢颖杰
用户价值-服务	服务提升 降30%电话订单比	自助变更取消优化二期	需求中	刘彦玮
用户价值-服务	服务提升 降30%电话订单比	担保订单提示优化	需求中	李钦
用户价值-服务	服务提升 降30%电话订单比	担保类问题标签	需求中	卢颖杰
用户价值-服务	服务提升 降30%电话订单比	在线服务配置“补开发票”touch页	需求中	卢颖杰、欧阳逸舒、刘彦玮
用户价值-服务	服务提升 降30%电话订单比	酒店发票物流查询	需求中	卢颖杰
用户价值-服务	服务提升 降30%电话订单比	订单详情页状态条“已离店”后子节点优化	需求中	刘彦玮
用户价值-服务	服务提升 降30%电话订单比	优化发票抬头填写及保存功能	需求中	刘彦玮、李钦
用户价值-服务	服务提升 降30%电话订单比	自动开发票功能	需求中	欧阳逸舒、卢颖杰
用户价值-服务	服务提升 降30%电话订单比	数据分析	需求中	杨准、欧阳逸舒、余燕颖、李钦、陶淘

注：579 战役项目 80%的目标已经达到，其他项目目标由于后期（C&Q）整合，进行了目标调整。

机票和酒店部门的经验对于创新产品的演化有着深刻的指导意义，例如火车票、度假、门票、团购以及当地人业务，基本是学习现有的模型，从覆盖和低价做起，复用现有搜索和 TTS 的模式，逐步建立产品核心优势，随后将重心逐渐调整到用户体验和用户服务方向，在此过程中不停地通过战略战术的调整去快速试错，追逐一个又一个的高峰。当然不同的业务和产品存在很大的差异，例如火车票天生不具备价格差异，那么基本就无法通过低价来俘获用户，更多从用户体验和服务来实现用户价值，如引入纸质票、抢票功能等。

对于业务团队，价值模型又是如何运作落地的呢？

这就要说到公司的产品目标分解。目标分解工作由部门负责人牵头，根据价值模型的树形结构，结合团队成员的意见，划定业务重点以及各分支的预期目标，再将各分支目标分解，如将用户服务的出票服务量化为第一季度实现出票时长小于 30 分钟，再进一步规划通过哪些项目来实现这一指标，同时明确所需资源。产品人员在了解并认可部门目标的前提下，规划自身目标以及工作内容，每一个成员都为达成部门目标而努力。项目执行过程中，前评估数据可以帮助纠正方向，后评估数据则可以体现项目价值，验证战略方向的准确性，反作用于价值模型的调整和目标的制订。如此形成闭环控制系统，引导团队和个人不断实现团队价值和自我价值。

举个典型的例子，2013 年年初机票部门成立了 8 个专项项目。这 8 个项目是当年或当季的各部门的重点，对应当时价值模型上的高优先级和重点，每个项目下面会有具体的需求点和策略，并落地成一个个小项目。所有需求点并不是一开始就全部定好的，而是根据上线后的即时效果（后评估）来不断优化和矫正。

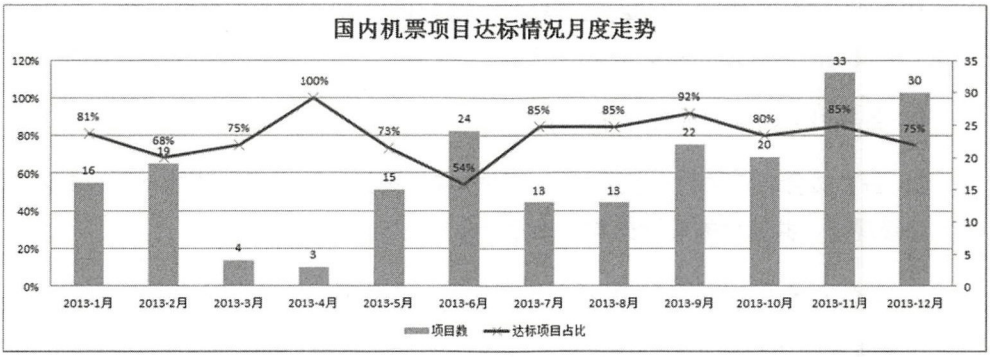
Level1	Level2	Level3	level4	level5	KPI	策略&任务	
用户层面	用户安全和服务有保障	服务监控和保障					
		资金、信息安全					
	用户体验好	数据准确					
		快捷便利					
		信息明确					
		全场景覆盖					
		用户粘度高					
	产品质量好	产品竞争力强					
		产品丰富					
客户层面	客户运营顺畅、效率高	提升代理商运营效率					
		保险运营					
		支付运营					
Qunar自身	内部管理与流程完善	基础信息维护、检查与发布					
		内外部工作流转顺畅、有据可依且有时间节点控制					
		流程控制和管理					
	风险控制	加价风险控制					
		行程单风险					
		发票风险					
		供应链风险					

注：机票 2013 年目标前 3 级拆分。

项目	已完成	取消	暂停	总计
Project1-资金安全	5			5
Project2-多报价	4	1		5
Project3-服务体系和用户体验	6	1	1	8
Project4-可靠低价	4	5	1	10
Project5-准确率和实时性	1			1
Project6-TicketTime	2			2
Project7-旗舰店统一	13		2	15
Project8-代理商运营服务平台	2	2		4
总计	**37**	**9**	**4**	**50**

注：机票 2013 年的 8 个大项目。

PMO 组织团队通过后评估（项目过程和效果总结）的方式来实现闭环，后评估周期不固定，会根据项目和其所需要的运营周期来决定，一般会在上线后当天、三天、两周、一个月，甚至半年时去观测数据变化。要求评估方法在立项时即明确，并且前评估和后评估方法必须一致。以此实现目标→执行→结果→新目标，达到闭环。



注：2013 年国内机票项目达标情况，全年平均达标率为 79%。

随着当下产品迭代的速度越来越快，对于管理者的素质以及团队的响应速度要求也越来越高，公司员工不仅经受住了考验，反而做得更加出色，我们通过不断调整价值体系，快速响应市场需求。

6.2.2　委员会机制

“去中心化”委员会机制

凯文·凯利，最受关注的未来学家，在 1994 年他的《失控》中，远见并大胆地

“预言”了人类的“未来”。20 年后的今天，书中的“预言”被发现是正确的，并且对我们可预见的未来，依然具有重要的指导意义。其中表现最为显著的是，组织的协作复杂化和去中心化。

携程从建立到现在的 17 年，自身由小变大，研发团队人数和组织越来越多，自身的产品协作复杂度成指数级增长。携程采取的应对措施与大多数大规模组织相同，进行了事业部化的拆分。将原来的按工作产品性质划分的中心化部分，拆分到各个事业部。

事业部化的拆分毫无疑问更有利于各事业部的独立快速发展，且事业部内部的沟通协作效率也大幅提升。但同样带来了去中心化造成的事业部间的互相协作问题，一旦需要做协同的产品研发工作，进度就很容易受到影响，还出现了每个事业部都在造“车轮”的问题，如“车轮”样式和性能的不统一等一系列的问题。

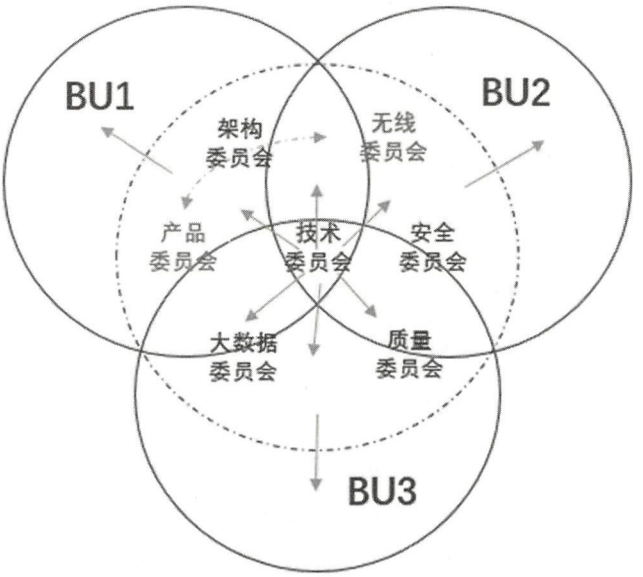
在这样的背景下，委员会这种发挥 BU 串联管理功能的制度就应运而生了。当出现跨 BU 产品的研发需求时，委员会就会发挥跨 BU 的协同效果。携程这种“去中心”的委员会机制的实践模式和更常见的“中心级 PM”不同，这种不同主要表现在：

1．各委员会内部的成员并非汇报关系。

2．各委员会各自关注一块领域，在该领域内更关注跨组织的协同，通过共同制订公司级规范、共享优秀设计架构，串接跨多个 BU 的业务逻辑。

3．委员会采用主席轮值制和决议投票制等民主模式，构建平等的虚拟组织文化。

以携程委员会模式为例，质量委员会成员主要是各 BU 的测试领导，委员会主要关注各事业部的测试和软件质量。而架构委员会是由各 BU 的架构师组成的，定期会议着重讨论公司的技术规范设定，交流各种业务场景的架构设计，为业务 10 倍增速做好技术的铺垫工作。即便处于下图中核心位置的技术委员会，也是为了更好地交流沟通各委员会的工作而设立的，大部分成员也来自于各委员会，通过定期汇报来充分地交流各种信息。一些重要的信息也通过这种类似波浪的方式散播开去，很好地化解了事业部之间的沟通壁垒。



因为委员会成员来自于各个 BU，定期的交流会帮助大家更好地获取有价值的信息，很好地避免了“造同样的车轮，或者造车轮标准不同”的问题，并且还会带来巨大的业务价值，这样的例子很多。

例子 1：携程大数据委员会关注的大数据，会串接同一个用户在购买机票（机票事业部）、入住酒店（酒店事业部）、挑选度假产品（度假事业部）的各种行为数据，更全面地勾勒出一幅用户画像。这对后期业务团队定向地推送携程旅游产品信息有很大的导向性，订单转化率也有一定的提升。

例子 2：携程架构委员会则会采用民主的投票制度来确立公司级的公共规范，比如，框架、CIS 等公共团队都引用了一些第三方组件，而很多第三方组件在跨越大版本后会有不兼容的问题。这种情况下，架构委员会就发挥了重要的作用，统一了公共团队所用的第三方组件版本，避免版本冲突。同时，一些 BU 因为更深入地了解了其他 BU 业务场景，充分吸取了其中适合自己的架构设计思路，比如携程玩乐团队和度假团队在业务场景上有一定的类似性，两者的交流就会带来很大的价值。

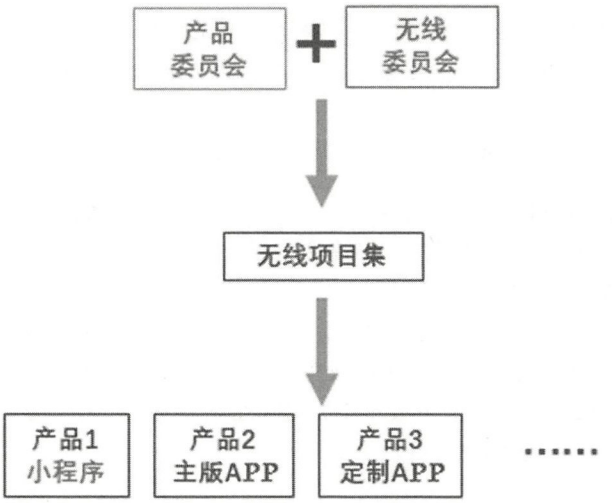
由于委员会自身涉及了多个业务团队，如何构建和运营委员会呢？这时候技术中心的 PMO 价值就开始体现了。因为随着项目管理从传统的瀑布方式转向更为灵活有效的敏捷方式，项目经理职位在慢慢消失，产品经理慢慢取代了一部分的项目管理职能。新型的 PMO 在继续履行一些类似制订公司级项目管理流程规范的事务外，在这

种“去中心”的运作模式中扮演了重要的串接角色。

比如携程技术中心的 PMO 会为每个委员会配置委员会秘书，这些秘书通常都是经验丰富的项目集经理，他们本身就因为长时间推动项目集工作而非常了解各个业务线的核心人员，通过 PMO 的驱动，可以快速地组建出各个委员会的核心成员，然后由这些核心成员一起讨论和界定委员会的专业领域和主要职责，再由委员会秘书协调完成最终的委员会人员组成。

PMO 的这些项目集经理还会负责各个委员会的日常运作，最大限度地把非技术类事务处理完毕，帮助委员会的轮值主席确定委员会的阶段性工作内容，并产生和驱动一些重要项目，这类项目可能是技改类项目，也可能是业务相关项目。

上面所谈及的委员会中，与产品最为息息相关的就是产品委员会，这个委员会中除部分技术人员外，有产品管理的人员，还有很多业务线的代表。大家通过对业务、产品和技术的讨论，会把跨业务的产品需求定义得很清晰，同时也自然地确定了优先级。这很好地串接了业务、产品和技术，相应的同事也会把这些清晰的需求带给各个项目集，而项目集负责人则可以驱动后续一系列关联性的项目，向着正确的产品方向迈进。



6.2.3　履带式行走

携程经历了 17 年的快速成长，目前已成为业务多样、设计复杂、规模庞大的“坦克级”体量的大型公司。大家都知道，坦克很重，如果改用车轮，那么车轮与地面的接触面积就会很小，开在田野里很容易陷进去。坦克安上履带，轮子在履带里滚动，遇到沙地、雪地和泥地，宽宽的履带把坦克的重量分散了，车轮就像走在路上一样方便。所以携程采用了“履带式”行走的方式。携程的委员会模式相当于这根履带，对各业务起到了“牵引”和“保护”的作用。

履带的“牵引”是如何玩转的呢？各业务自行发展通常会有着各类“封闭式”集团的通病，即信息不对称、重复“造轮子”。委员会提供了这样一个平台，从产品、架构、无线、大数据和质量方面横向地“牵引”业务的发展，维持在相对统一的高度，互通有无，协同合作。

履带的“保护”是如何起到作用的呢？业务间因自身的利益不同，思考问题的出发点也不一致。在执行跨业务的“公益活动”的时候必然存在和自身利益冲突的可能。委员会提供了一个沟通和协调的机会，让更多跨业务的“公益活动”有施展的舞台。

说了那么多，让我就以时下最流行的微信小程序为例。谈谈携程是如何进行“履带式”行走的。小程序作为微信战略布局，随着相关入口的开放，其所带来的影响必然巨大。自 2016 年 9 月 22 日，微信对外宣传小程序已经出来并开放了 200 个内侧名额，携程就积极加入了小程序的“开荒之旅”。

在初创期，各种因素都在变化，上线时间不确定，审核标准不明确，使得携程小程序的开发举步维艰。这使得小程序的协调和沟通工作变得至关重要。在不明确最终上线时间的前提下，我们采用 Scrum of Scrums 的方式，两周一个冲刺，迭代开发。在多次提交微信审核未通过后，反复修复问题，增加新功能。确保在上线前给出一个功能强、体验好、质量高的产品。而携程的公共团队，基础、市场、框架、支付也配合业务的需求，和微信给出的新功能和新特性，以敏捷的方式为各业务提供保障。以每两周一个迭代的节奏，配合微信内测环境的不断更新，将可交付的功能提供给业务测试。也就是说，公共团队内部，公共团队和业务的配合，携程整个小程序团队，以及小程序团队和微信的配合，都是以敏捷的方式两周一个冲刺的节奏稳步前进。对于处于初创期的小程序来说，一切都是可变的，如何在可变的环境中掌握同步的节奏，是小程序制胜的法宝。

小程序的 size 限制、快速整装、迅速上线对携程这样一个“巨型坦克”是一大障碍。哪些业务适合加入小程序，可以分配的 size 是多少，直接决定了该业务产品的体验。而这部分的定位，对于“从未吃过螃蟹”的我们，显得相当的模糊。对于首次加入业务是需要勇气的，如何不辜负这份“勇气”，我们把这个艰巨的议题提到了产品委员会，由各业务产品经理一同商议业务的去留和规模，并制订了后续各业务分配 size 的规则。“履带式”行走，让携程这辆大坦克在面临新机遇时迅速决策，轻装上阵，快速突击。

刚刚度过“初创期”的小程序团队，初尝了小程序的甜蜜果实，新的需求纷至沓来。现有业务功能的完善、新业务的加盟，使得仅有的 2MB 容量捉襟见肘。为了尽可能满足业务的需求和最大化收益，产品委员会通过了业务上下线标准。使得更多适合“短、频、快”形态的业务能够融入到小程序这个大家庭。而为了配合小程序各业务上下线，一个月两次的常规发布，加上个别配合市场活动的紧急发布，为各业务需求的更新和新业务上线的尝试提供了便捷而有效的途径。

不仅仅是携程主小程序，各业务的独立小程序和附近小程序随着微信入口的逐步开放和市场的成熟应运而生。而进入“成熟期”的小程序群体，更需要的是规范化的发布流程和质量保障。此时，由市场、基础、框架和 PMO 成员组成的虚拟团队，根据携程小程序的现状，梳理了现有的产品、开发、测试等遇到的问题，以及根据市场的快速变化，制订了各项标准化流程、上下线规范以及告警监控方案。“履带式”行走，让携程小程序这个鬼灵精怪的“小怪兽”能够适应今天市场的不断变化，应对各类机遇和挑战。

微信小程序这个案例，让我们看到携程的这条履带具有较高的抗拉负荷、良好的韧性和耐磨性。让携程这辆巨型坦克，凭借六大委员会的独立运作，互相输送信息，相互牵引，互为依仗，在当今这个环境多样的市场环境中，进行“履带式”行走。

6.2.4 跨部门大产品的推动心得

2017 年年初，James 提出要开发一款新产品，即基于用户定位，区分行中异地和常居本地场景，进行不同的智能推荐，包括吃喝玩乐购等内容信息和商品推荐。主要目的不仅为用户提醒行前灵感、行中目的地信息推荐，同时还强化本地休闲旅游场景，满足越来越高频和碎片化的泛旅游类需求。

这款产品需要携程集团下几个事业部通力合作。首先是攻略社区提供目的地信息，然后是门票部门提供门票预订信息，玩乐部门需要提供当地玩乐信息，周边游部门提供旅游等信息。

在公司级的产品会上，大领导拍板由攻略社区来主导，所幸我也是其中一分子，协助推动来完成这个产品。在半年的跨部门产品沟通与推动中，我的感受心得如下。

首先，关于跨部门产品的部门间的合作问题。各个部门对目标是否认可？目标是否统一？目标不统一的话，大家的力道分散到各个方向，不能凝聚和发挥最大价值，甚至互相掣肘。

如何统一目标呢？主要就靠软能力，沟通协调，各部门主要产品负责人开会，统一认识和思想，目标明确化，同时会议邀请重要大领导参加，如果会议讨论不出结果，就让大领导拍板确定。

目标统一的话，则涉及分工问题，分工是否清晰，各部门大家各自的业务范围是什么？各自的责任是什么？权责要清晰。还有就是谁来主导？从人性角度讲，大家都要争主导权，一般按照切分到各个部门的目标重要程度以及责任大小来确定，但是很多时候即使弱势的部门责任更大，理应由其主导，但是强势部门会隐性掌控主导权，这点需要明确。

确定了主导部门后，谁来汇报更是一个严峻的问题。工作多年的人士都了解，在大集团公司，能在公司级的会议上汇报产品与项目，自然领导对汇报人的印象更深刻，汇报权带来的好处不言而喻，谁来汇报，是一个博弈的过程，需要动态平衡。

关于主导权，最可怕的是表面认可，内心未必真正的认可。碰到大产品，尤其是集团 CEO 关心的大产品，每个部门必将拼死争夺，如果被 CEO 指派到其他部门主导，需要提供支持与配合的部门自然心中不爽，不乐意积极配合，甚至使绊子，拖累产品的实施进展。

有人会问，碰到这种情况怎么办？向大领导反馈？一般情况下碰到沟通协调处理不了的问题，大家的惯性思维就是我搞不定，我找别人，找朋友，找同事来沟通，再搞不定，找领导，将问题升级，让高层去沟通，达成一致。这种在公司级别的会议上明确的事情，对方暗地里不配合，大领导未必不清楚，这种情况下再没法向上反馈了。只能是从商业合作的角度出发，大家从产品中各取所需，各自获得各自相应的利益，

确保收获与付出相匹配。比如 A 认为 A 好，B 认为 B 好，则 AB 合作，大家优势互补，互相配合。

待以上目标、主导、合作、汇报都明确后，具体任务的拆分、执行和配合则通过流程和沟通机制来保证。流程和沟通机制可以采用现有的，也可以大家集体开会再明确一套。

其次，关于跨部门产品的短期目标和长期目标的问题。甚至某些时候目标是矛盾的，如既要求流量，又要求订单量。这种情况下，需要进行目标分解与预期管理。

目标分解，分解为过程性的目标和终极目标。一款大产品总是由 *N* 个阶段，*N* 个项目组成的，如 A→A1→A2→A3→A4→B→B1→B2→C，C 的终极目标可以和 B 不一样，分解确定每个过程的目标后，能更加清晰地设计产品方向，有针对性地投入对应的资源。

产品负责人最头痛的问题应该是大领导的目标变动怎么办？今天说这个，明天说另一个。最常用的方法是加强沟通，和领导讲讲我们近期做了什么，准备做什么，但是集团公司的大领导就没这么多机会可沟通了，和大领导的沟通时间有限，董事局主席，就那么几分钟，说不了太多，这种情况下也要讲究动态平衡，想办法满足领导的部分需求，同时站在领导的角度考虑问题，努力完成。

再次，关于产品的目标达成测量问题，测量标准是什么？谁制订？谁测量？至关重要。这次这款公司级别的大产品就碰到了这个问题：项目阶段性完成后，需要评估阶段性的成果，结果基础一套数据，各个事业部一套数据，大家的数据都不相同，甚至某些点相去甚远。召集各个部门的数据组负责人一起开会沟通，才发现是口径不一致，算法不一致，于是又开无尽的会议来讨论以谁的口径为准……

总之，跨部门的大产品，要明确统一的东西太多太多，会议必不可少，各种机制要健全，流程要健全，才能最大限度地减少后期的种种不确定性因素。到 2017 年 9 月，在大家集体的努力下，我们的这款跨部门大产品——玩转当地，也问世与大家见面了，如下图所示。

景点・玩乐、周边酒店、周边游等也与“玩转当地”频道的精准的信息内容深度结合，极大缩短了用户的决策路径和时间，更加快速地响应用户需求。

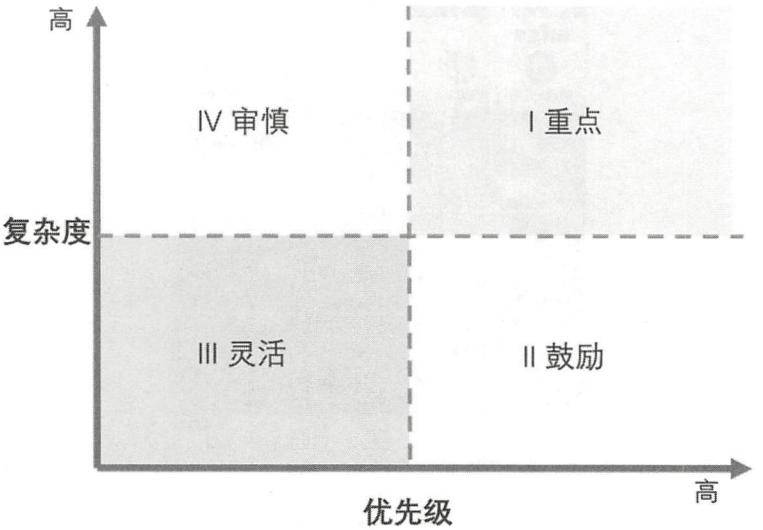
本/异地场景的区分是携程“玩转当地”频道最重要的特色，在未来，本/异地场景还将覆盖到“当地攻略”、“周边游”等更多频道，意味着这款产品将会是携程各个部门共同参与打造的一款精品。同时在大数据 AI 团队的支持下，未来携程“玩转当地”频道将以智能化的千人千面的面貌，成为每一位用户在无论旅行还是日常生活中都离不开的随行智能贴心导游。

成功的产品，伟大的产品离不开伟大的团队。大产品，小团队，各部门积极配合，大家都小步快跑，快速试错迭代，将会有很多的优秀的大产品诞生并倾情奉献给用户。

6.2.5　跨公司产品管理

随着携程在 OTA 领域的不断发展壮大，为巩固领域内的领先地位，开始对行业上下游和竞争对手进行并购，集团间公司的产品研发协作变得越来越密切。为了有效

管理跨公司产品的研发与运营，依据产品需求的复杂度与优先级属性，实施了不同的管理策略。



重点类——项目管理，重点跟踪

重点管理类产品，涉及公司间业务的重大重组，或是基础产品平台的融合建设。推进此类产品的研发与部署，往往需要投入较大的成本，并冒着较高的失败风险。

为了确保完成目标，管理方法一般是：成立跨公司组织的项目团队，以项目管理模式进行管理。项目由各事业部负责人发起，任命各公司的项目经理，并指定跨公司的项目协调人，共同组成项目管理团队，由项目管理团队管理和跟踪项目目标的达成。采取以上措施，有效促进了平台基础产品的整合统一，这类“修路”项目为公司间的业务协作铺平了道路。

鼓励类——鼓励协作，积极创新

跨公司业务合作产品占大多数，各公司事业部间的业务合作，大多由产品经理驱动。在这个类型的领域里，经常会出现一些合作创新类项目，如人工智能和大数据应用产品。

携程会通过鼓励政策，促进类似产品的合作发展。为此设立协作项目奖，奖励跨

公司间团队的协作。近两年，共有 23 个项目获奖，涉及各业务领域，产品直接收益近 20 亿元。

灵活类——适度轻量

大多为创新类合作产品，动用的资源和影响范围有限，由各组织内部决策孵化，然后邀请外部企业团队加入。集团根据此类产品项目的特点，制订了跨组织的业务成本交易原则。依据公平协作的交易原则，各公司组织间分担成本与收益。

审慎类——冻结观察，慎重决策

在较为充分的自组织团队中，想法往往多于资源，很难在诸多创新建议中做取舍。我们采取的基本原则是价值法：产品团队需要经过基于实验的收益评估计算，来确定产品预估收益，在与成本比较后，根据 ROI 来确定产品优先级。此方法与一般产品研发决策差异不大，但因跨公司间的协作，复杂程度会被严重放大，存在较大的成本风险。如果产品项目的优先级不高，并存在较高的复杂度，发起组织会将其冻结再评估，如果优先级没有提高，或复杂度没有降低，产品又不具备战略意义，所以并不建议启动此类产品项目。

传统企业以供应链方式进行关联，一般采取依据合同的严格供销关系管理。采取这种方法，可以有效降低不确定性，保证生产和商业的顺利开展。这种方法在需求相对确定、技术手段稳定的场景下，仍然是最为有效的方法，我们仍然推荐大家使用。

如前文所述，随着社会和技术的发展，人、组织、公司间的协作持续复杂化，需求与技术的不确定性越来越多。现在行业领先的企业采取的措施是首先整合生态链，进行上下游资源的并购，在行业中进行整合。但不同以往的并购，出于去中心化等多方面因素考虑，并购后大多不会进行深层次的融合重组，而是采取多品牌兼容的群狼战术。

现在的企业组织间的结构越来越复杂，而且还在向更为复杂的结构发展，这造成管理成本的大幅提升。根据需求的“优先级+复杂度”两个属性维度，视不同情况采取不同策略，既能降低成本，也可充分激发组织间的创新能力，达到“严管”+“搞活”的最佳效果。

6.3 创新驱动产品迭代升级

大产品经过精心耕耘后，产品逐渐成熟进入运营和维护阶段，此时要想再出大招就离不开产品创新。创新又离不开组织的土壤，携程的创新工作坊和创新工场会让你大开眼界，相信我们的理念创新、创新产品迭代升级之路和创新项目如何从创意到落地实践，能给你带来一些启发。

在很多人眼中创造一个全新的事物才能叫创新，也有一部分人认为做出小小的改变也是创新。其实这两者都没有错，创新本来就分为突破式创新和渐进式创新。就我们所处的旅游行业而言，大部分业务是依托交通等其他产业的一种延伸，所以在旅游行业本身上实现突破式创新，目前还是很有难度的。

6.3.1 创新工场——创新项目从创意到落地

创新工场自 2016 年 6 月创立以来，紧跟集团创新创业的战略目标和方向，秉承着激发公司创新氛围和培养创新人才的原则，为创意提供落地的平台，也为员工提供更多的成长和发展空间。

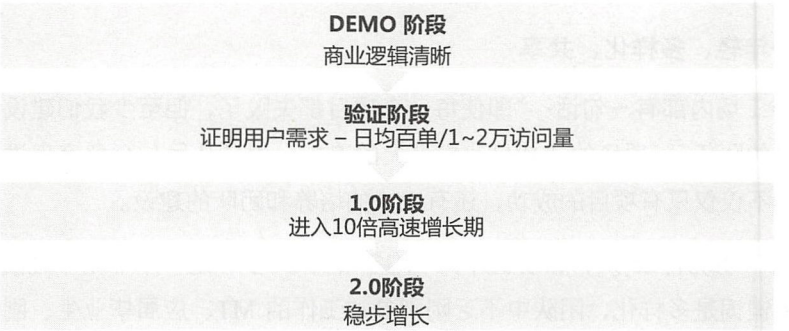
平台——探索、实践、跨部门

在过去的 1 年中，创新工场花了大量的时间进行了创新方向的探索和创新项目的实践。

创新探索方面，我们收集了约 200 个来自携程各个 BU 员工的创意，并与每一个项目申请者进行了至少 1 次的深入沟通。若遇到可行的创意，初次沟通之后，创新工场会邀请各 BU 专业评审人员对项目及其商业模式进行 1~2 轮的评审。与此同时，我们进行了大力度的宣传，让携程人知道有这么一个地方可以释放他们的创新激情，通过 Ctripall 邮件和微信公众号与全公司的员工分享我们的探索成果，同时邀请了在热度较高的创新领域的专家与我们的项目申请者进行面对面的交流，加深其对于创意的思考和完善。

创新实践方面，自 2016 年 8 月份开始尝试第一个项目——机场停车之后，陆续上线了 10 个项目。在项目实践过程中，创新工场总结出了四阶段模型（如下图所示）

和 B 点模型，每一个项目都在这两个模型的指导下进行创新。



在四阶段模型中，从一个创意到形成商业模式较为清晰的 DEMO，项目的产品逻辑和商业模式会经历不断的推敲打磨，创新工场会邀请相关领域的专家和大牛对项目进行轮番质疑和剖析，直到形成一个逻辑清晰的 DEMO，并放到创新宫格上开始它的表演。在验证阶段，创新宫格释放流量，为 DEMO 做用户需求验证，在每天几千 UV 的冲击下，产品的逻辑、场景以及用户需求将会被快速地验证。

达到既定目标（如日均百单或 1~2 万的访问量），则项目需求被认为验证成功，进入 1.0 阶段。1.0 阶段的产品需要找到最合适的场景和关键流量入口，提高产品曝光量和用户知晓度，达到 10 倍快速增长目标，进入比较稳定的 2.0 阶段。2.0 阶段的产品进入稳步增长阶段，需要做的是持续地优化产品和用户体验，并扩大市场份额。不同的项目在不同的阶段努力耕耘，以期望能达到既定目标，进入下一阶段。其中也有一些项目由于各种原因遭遇挫折甚至不幸夭折。

除了做 0 到 1 的项目，创新工场还扮演着另一个挑战者的角色。如果一个项目在 BU 内进展较为缓慢，而此时挑战者认为换一种方式能做得更好，他也能到创新工场来，提出自己不同的观点和角度，称之为“B 点”，AB 实验的 B。B 点模型更看重项目区别于原有模式的模式和方法，而这些模式、方法由于某些原因是相对不容易被复制的。通过科学的 AB 实验，最终验证 B 点是否更为有效率。B 点模式的存在对现有 BU 的项目起着督促的作用，时时促进 BU 进行内部创新。

另外，大家都比较关心的项目的归属问题其实并不是个问题。在创新工场，项目归属是一个很开放的命题，按照最适合项目发展的方向决定其归属。相对比较独立的大项目可能最终会成长为 EU，甚至 SBU，需要依赖于现有某些 BU 的延伸性项目进入运营阶段后，最终可能会归属于该 BU，因为种种原因失败了的项目也会面临着被

关停的命运。

团队——年轻、多样化、共享

创新工场内部有一句话："即使每一个项目都失败了，但至少我们建设了一个战斗力很强的队伍。"项目的成功是我们最大的追求，但是从另一个角度来讲，创新工场的成功不仅仅只有项目的成功，还有人才的培养和团队的建设。

创新工场现有 25 人，除场长外，平均年龄为 26 岁，是一个非常年轻的团队。另外一个关键词是多样化，团队中不乏刚刚参加工作的 MT、应届毕业生、刚刚工作一两年的职场新人，也有拥有相当丰富的旅游行业知识和经验的携程老人。30%的团队成员拥有国际化背景，足迹遍布美国、英国、芬兰和日本。更值得一提的是，创新工场团队成员来自携程各个 BU，曾任职于火车票、地面玩乐、攻略社区、地面交通、客服等事业部，下图为创新工场的大家庭。

不同的背景使得创新工场内部的分享学习氛围特别浓厚，一走进工场的办公区域，就听到叽叽喳喳亦欢乐亦严肃甚至争吵之声，每一次的相互交流都是促进彼此成长的一次机会。有很多渐进式的创新在我们身边不停上演，行李寄送项目也算是其中一个。下面让我们看一下创新项目——行李寄送，是如何从创意到落地再到实践的。

行李寄送业务的外部环境

行李寄送这个概念应该最早形成于美国，随后在日本和新加坡这个概念也慢慢深入人心。在这三个国家，行李寄送业务仿佛已经成为大多数人出行的一种必备服务，也就是形成了消费习惯。人们在出行前将行李寄到机场，到达目的地将行李寄到酒店，一路上都不用提行李。在美国这个服务更是被做到极致，航空公司上门办理登机手续，拿走行李，乘客到机场直接过安检就可以，到达也不用去转盘拿行李，直接去玩或者去酒店，行李会在第一时间被送到酒店。

而在中国，这个业务的起步从 2013 年左右才被人提出，然后一波创业公司开始模仿美国的公司在中国做这个业务，殊不知国情的不同，对业务的影响太大，最终因为不能和航空公司达成合作，导致公司经营出现问题，最终关门。一味地模仿抄袭并没有让这个业务在中国一炮而红，正因为这样，后面的公司学聪明了，我们不和航空公司合作，只帮客人送到机场或火车站，于是中国的行李寄送模式就这样形成了。

行李寄送在携程的成长

1．创意出现

这个业务在携程最早并不是我提出的，而是 MT 的小伙伴来创新工场提出了这个痛点，但是在第一次讨论中这个创意因为成本、安全、时效等问题被拍死了。当所有人都以为这个项目也就这样被过掉的时候，不撞南墙不回头的我并没有死心，在收集了一大堆资料，并且找到了当时提供行李寄送服务的 6 家公司，向老板说明了这个业务和现在市场情况，并解释了如何解决成本、安全、时效问题之后，终于让携程成为第一家开通行李寄送服务的 OTA。

2．验证需求

行李寄送在业务形态上其实就是取代现有的传统寄存业务，让旅客可以退房不用把行李寄存酒店再折返回酒店拿行李，或者早上到达后不用专门去酒店放行李。业务为旅客带来的不仅仅是不用拿行李，没那么累了，或者省了回酒店的打车费，最重要的是为旅客节约出了半天甚至一天的游玩时间，在时间成本日益升高的今天，这个服务确实是“用钱买到游玩的时间”。

正因为是一个全新的业务，大家都想知道到底有没有市场，所以一开始，行李寄送以一张图片和一个预订电话为原型图，在首页上线行李寄送宫格，而在没有任何团队的情况下，我兼职起了客服，所有客服电话会转接到我的手机上，那段时间，我 24 小时在接听客户的咨询或者通过手机要下单的需求。在锻炼了一身一线客服本领后，终于得出结论：这个业务是有需求的！正式立项，现在开始做，快速上线。

3．项目立项上线

正式立项之后，我的第一个小伙伴加入了，用 OK 制来解释的话，就是我这个 O 终于等到了我的 K。后端人员开始搭建产品后台，而我要开始设计产品原型。一个全新的业务交给我。开始我也不知道怎么办，曾经也动过心思去抄一抄别人是怎么做的，无奈我们是第一个做这个业务的 OTA，又能抄谁的呢？供应商为自己的公众号？不行，实在粗糙而且不符合携程产品风格，在一次次画的原型图被嘲笑之后，在 UED 的美化下，终于第一版设计出来了。这个时候，第三个前端开发小伙伴加入了我们的团队，团队的基本配置也算齐全了。

一个全新的业务，全新的项目，在敏捷开发的指导下，我们一共用了两周多的时间，项目在 2016 年 11 月 1 日正式上线。

4．项目成长

项目立项之后，有了自己的第一个目标，实现日均 100 件行李。为了这个目标，前后两个有着同样目标的商拓小伙伴加入到团队，业务覆盖的城市不断扩展，从上线最初的 8 个境内城市，2 个境外城市，到后面的 20 个境内城市，10 个境外城市，行李寄送服务正在为更多的旅客提供着便利，让更多的旅客认识和了解到这个服务。在项目组小伙伴的一起努力下，通过线上产品的改进和与其他 BU 的合作，还有和线下供应商及酒店的合作推广，在 2017 年 2 月，项目组达成了第一个小目标，日均 100 件行李。目前，项目组正在朝着第二个目标——10 倍增长，继续努力着。

我在这个过程中的成长

创业很难，创业失败的很多，我觉得可能创新+创业就更难了。一个全新的事物，怎么让旅客知道，怎么快速地让市场接受，都是在整个过程中我一直在思考和探索的问题。在这个过程中，如何合理运用现有的资源，也是创新和创业过程中的一个大难题。其实我们和其他很多在外创业的公司都会面对资源有限的问题，怎么来解决就只

能各显神通了。举个小例子，我们没有地推人员，也没有钱去找那么多地推人员，那就找供应商和我们一起来推广，提高佣金，让供应商出人，一起把这个新业务在线下铺开。很多事情总会有解决办法，只是看从什么角度来思考。

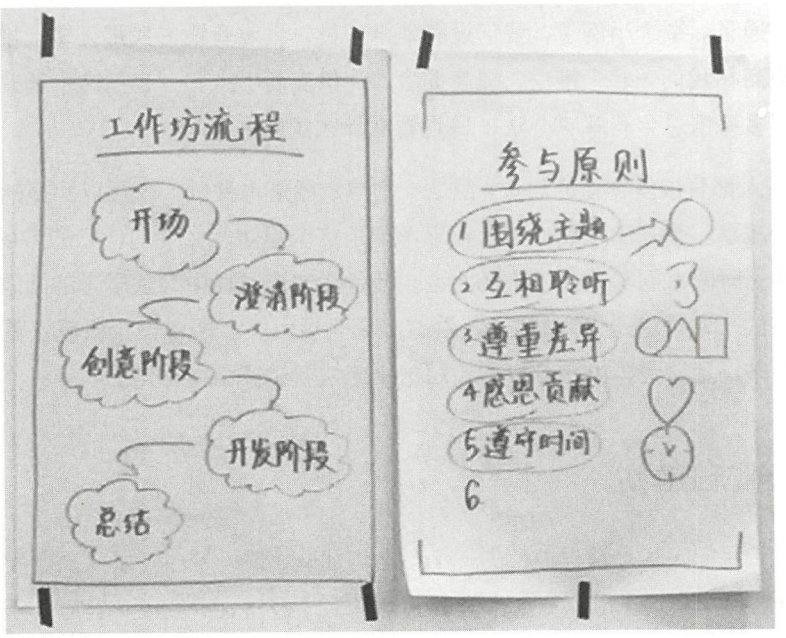
很多人都有很多创新的想法，但是一个想法能落地变成一个可以实施的项目却又是另一个故事。想法出现后，自己先思考一下这个想法怎么落地，它的市场有多大，我能在多长时间拿到其中多少市场，这些都是在这个过程中必须思考的问题，简单引用陈厂长的一句话就是："逻辑通，看看市场有多大。"。所以一个新想法首先逻辑要通，然后市场足够大，ROI 能打平，那这个想法就确实值得一试。

6.3.2　创新工作坊

案例：民宿团队是携程内部的一个创业型团队，最初是由开发人员自发组织的小团队，针对目前市场上刚刚兴起的非标住宿和共享经济，开发出一套服务于房东和房客的民宿业务平台。以非标住宿为核心的共享经济模式，平台引导社交沟通与信任感受，以及提供保障服务，撮合双方达成住宿意愿。

在这样一个初创型团队里，每个成员都可以是需求的提出者，搭建系统架构，完善功能，提出新需求，每个迭代中近 50%的需求来源于开发者。面对 Airbnb、小猪短租等竞争对手在国内的迅猛发展，如何打造出更强大的产品与之抗衡，是这个团队面临的主要问题。团队负责人找到工作坊的张老师，让她帮忙组织一个创新工作坊，带领团队一起获取更多的产品创新灵感。

怎样才能激发团队创意，探索出更加多样化的产品，提高产品竞争力呢？让我们一起走进民宿团队的创新工作坊，看看他们如何提炼出新颖的创意。创新工作坊的所有参与人员包括产品经理、开发人员和测试人员，由主持人负责引导所有的人员完成每个环节。



参与原则

1．围绕主题

2．互相聆听

3．尊重差异

4．感恩贡献

5．遵守时间

开场

开场由一场采访开始，主持人选择了 5 个主题，每个小组围绕一个主题对所有的成员进行采访，包括：

- 3 分钟计划
- 3 分钟采访
- 3 分钟总结
- 3 分钟展示

采访的主题可以是当前经济和社会的发展趋势以及本行业相关的发展趋势等，包括：

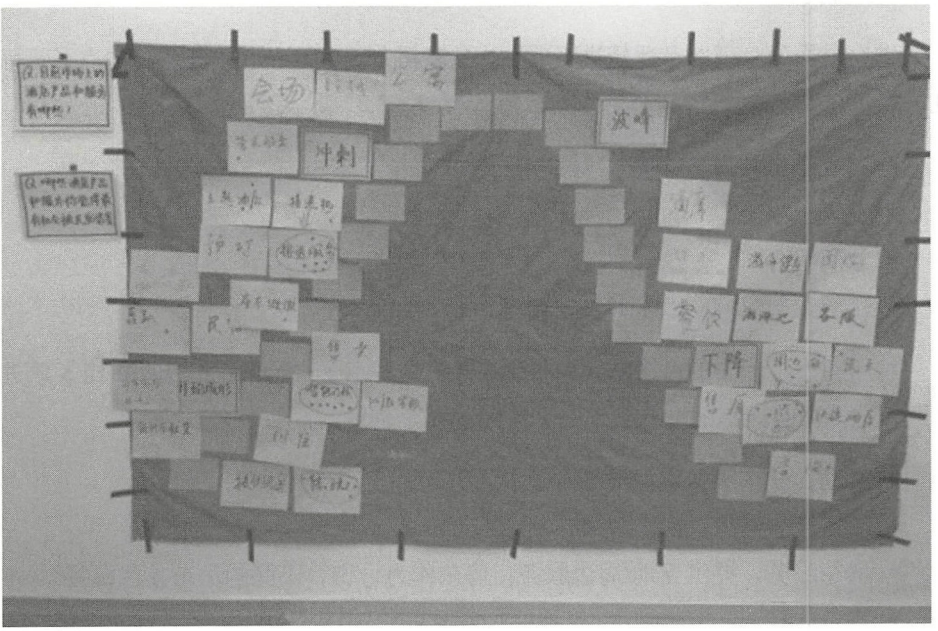
- 经济发展趋势
- 旅游行业发展趋势
- 技术发展趋势
- 社会发展趋势
- 机会和威胁

澄清阶段

1. 目前市场上的酒店产品和服务有哪些

列出所有的产品，并且把它们归类到不同的发展阶段，如下图所示。

- 上升阶段
- 冲刺
- 波峰
- 下降



2．哪些酒店产品和服务你觉得最有机会被借鉴

所有的团队成员投票，选择最多的几个进行归纳和分类。

3．脑洞大开

每个小组设定一个主题，小组成员贡献创意，组长不变，组员轮转，一起来完善。

- 客户的期望是什么
- 什么会使客户不高兴
- 你喜欢鲸鱼的什么特点
- 什么会惹怒客户
- 竞品哪些方面做得很好

通过对这些问题的分析，以及参与者的群策群力，可以碰撞出更多的火花。

创意阶段

- 记忆
- 逻辑联想
- 创新

结合前面的分析以及逻辑进行联系，来获得创新的点子，收集各组的点子，可以进行点子交换。

- 独立思考并写下答案
- 三人一组分享
- 重新组合
- 小组选出最好的点子并加以整理

将所有小组选出的点子贴到白板上，并对这些点子进行归纳分组，每组 5 票进行投票，选择票数最高的进入到下一个阶段的 4 个创意。

- 互动，房东服务意识培养
- 丰富个人信息，房客故事
- 房东服务，提供当地/周边服务；房东作为导游，不同的房东特色标签
- 安全体系

开发阶段

各个小组成员针对每个创意，形成完整的解决方案，并且选定组长向所有的参与者阐述方案，所有成员对这个方案进行打分，打分规则分三个方面进行：

- 有趣
- 新颖
- 可行性

总结

主持人针对整个流程做归纳陈述，团队成员们也获得了 4 个创新点以及完整的解决方案，后续会推动这些方案逐步实施落地。

在随后的三个月内，民宿团队针对上述 4 个方面进行了开发和上线运营，例如增加了房东信息的展示，将房东的年龄、职业、星座、爱好等信息展示给用户。对于房东的预订成功率、确认时长、超时率等信息进行计算，展示给用户并且应用到排序规则里面，一方面提高房东的服务意识，另一方面也更有利于提高用户体验。在安全体系方面，通过对接网安系统，进行入住人身份证的验证，并将信息传递给房东，有效地确保了安全性。经过几个月的上线运营，订单量与增长率均有了大幅的增长。

如果你的团队遇到了民宿同样的问题，不妨也来一次创新工作坊，开拓团队思路，激发团队创意，探索出创新的、多样化的产品。

6.3.3　创新产品持续迭代之路

> “自己在携程的二次创业，就是一个围绕‘创新’进行的变革，只有不断保持创新的精神和能力，才能在市场的激烈竞争中立于不败之地。”
>
> ——梁建章

“小老虎”模式下整合资源再创业

在携程从 2012 年下半年开始的变革中，组织机构的机制调整被摆在第一位。在各大事业部的基础上，把公司的权力分配，重新从总部分下去，各事业部下也成立了小团队，拥有更多的自主权、能动性、容错空间去快速推出创新产品，这种模式在携

程内部被称为“小老虎”。“小老虎”模式给了各个事业部更大自主权的同时也给携程的内部创新注入了无限的活力。在携程“小老虎”模式下，团队领导有充分的自主权可以灵活调整团队各个业务线的资源投入比例，按照自己的意愿进行布局，以充分利用资源做更多的创新尝试。

风口上的“共享经济”

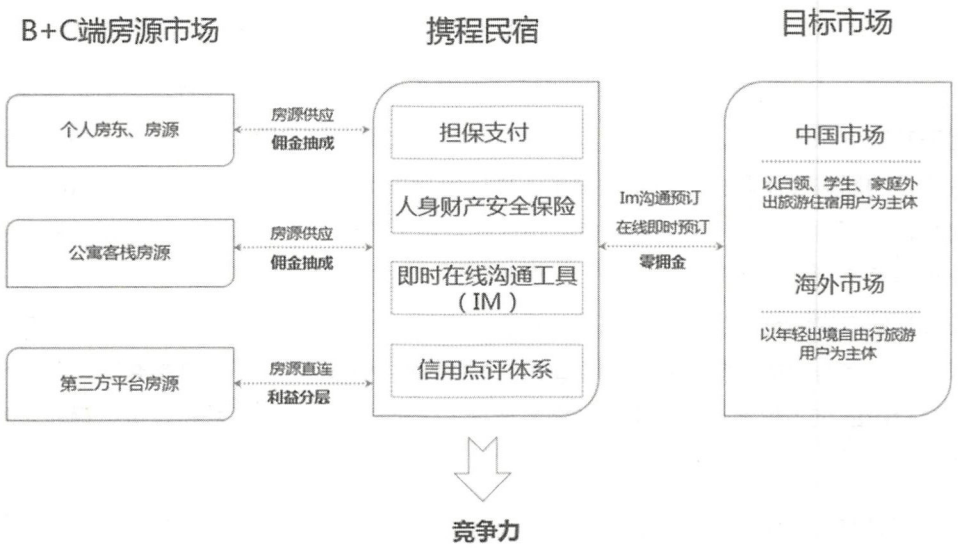
2014 年滴滴和快滴两家出行方面的公司之间的补贴大战引起了大家对“共享经济”模式的极大关注。除了“行”，“住”方面的共享经济代表 Airbnb 也在同年 4 月份宣布获得 D 轮 4.5 亿美元的融资，公司估值从 25 亿美元猛增到 100 亿美元，对传统酒店业行形成巨大的冲击。

再看国内市场，在线短租平台大体分为了两类，一类是以途家为代表的“B2C”模式，一类是以“小猪短租”为代表的 C2C 模式，还没有大的巨头玩家介入。对于创业来说，这样的局面无疑是个好机会。

内部创新的思考

携程作为旅游行业垂直领域的佼佼者，拥有巨大的流量入口和优质的会员，内部创新的话选择旅游相关服务更容易切入，还可以与携程现有的旅游产品形成合力，共同服务好用户的旅游全流程。

当前国内酒店行业的发展已日趋饱和，非标住宿逐渐成为有益的补充，对 OTA 来说无疑是一个不可忽略的潜在增长点。但民宿区别于酒店，包括民宿的业态、预订流程等。民宿是以非标住宿为核心的共享经济模式，面对房客应该向用户提供更有意思的房源，更符合用户多样化的住房需求，更能满足当下消费主力军对于个性化的追求；面对房东应该把房东闲置的房屋资源线上化，降低房东集客、房客搜寻的难度，为房东提供完善的房源保障与服务。另外，平台要提供评价体系，解决双方的信任问题，引导社交沟通与信任感受，以及提供双方保障服务，撮合双方最终达成住宿意愿。



携程民宿在这样的大环境下开始在携程内部进行孵化。

持续迭代之路

1. 技术驱动，市场摸底，4 个月内 MVP 毕业，孵化成为民宿业务线

项目初期其实大家对于民宿业务整体的现状并没有直观的数据感受 ，为了能更好地了解当前市场的整体现状（民宿到底是不是风口？是的话，风有多大？目前市场上的主要玩家过得怎么样？ ），急需要做一次市场摸牌。

大家首先想到的方式就是做市场调研，但目前整个团队能抽调出来的基本都是技术资源，程序员们可能无法承受这个负担。“我只想安安静静地做个写代码的美男子，让我走出去抛头露面？这样的需求不接！”此外，有限的财力、资源也不允许我们依赖于第三方的市场咨询服务。但正是这群可爱的程序员们神奇地用最低的成本实现了利益的最大化（只要需求明确），其实技术的边界远比我们想象得大得多。

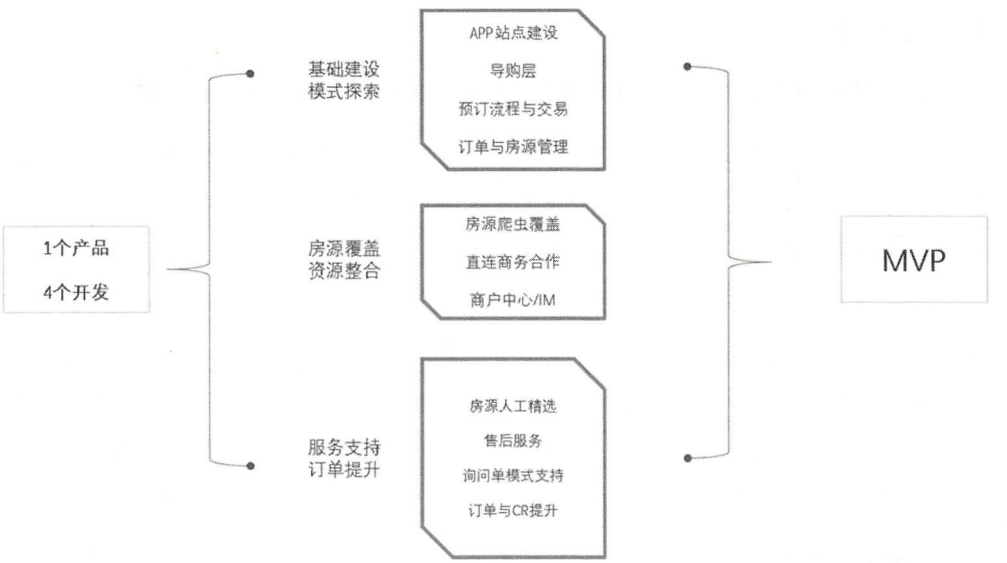
爬虫竞品分析

当前的互联网时代，信息越来越透明公开。竞品网站上真实的商品包含的信息其实很多，多到足够我们获取和分析到我们想要的各种数据。而且这些数据比一般市场调研得到的更加全面和准确，因为真实的用户选择行为才是最有价值的分析数据源。但前提是要建立起正确的数学模型，比如通过一定时期的全站房源库存变化推算每天

的房源订单量，通过特殊处理的房屋编号获取总的房源数量以及变化趋势等。

从另一个方面看，爬虫数据分析看起来很美好，成本低、可靠性高。但实际操作起来其实没那么顺利，每一次的数据抓取都是一次爬虫和反爬虫之间的技术较量。比较成熟的网站在反爬虫方面做了很多工作，包括关键信息图片化、限制同一 IP 访问次数，甚至对爬虫程序“投毒”等。错误的数据源不仅起不到数据分析作用，还会让分析结果产生严重的干扰和误导，这需要合理的数学模型进行数据的合法性校验以及脏数据的二次清理。虽然整体上困难重重，但经过一段时间技术的攻防，我们还是获取到了大量有用的数据，这对我们后续介入到民宿业务起到了有力的数据支撑。此外，这个阶段的技术积累也为后来自己做反爬虫、防止竞争对手恶意数据抓取积攒了宝贵的经验。

就这样，创业初期，我们通过技术创新的手段在没有任何市场和地推人员介入的情况下完成了一次完美的市场摸底。不仅节省了大量的调研资金，也锻炼了团队的技术。



自营还是平台

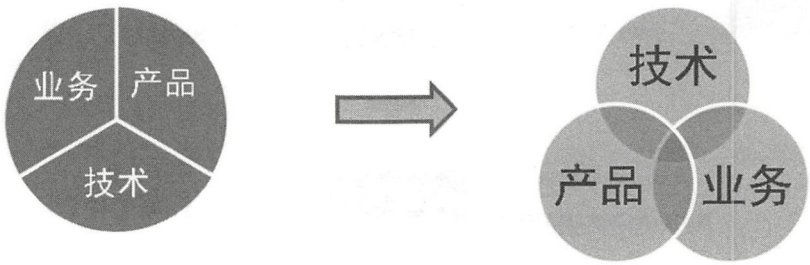
在确定了业务方向可行以后，下面还面临一个方向性抉择，做自营还是做平台。做自营能更好地控制上游房源质量、房东服务标准，能按照携程的标准制订退订政策等，但同时自营需要大量的线下人员进行接入推广、房屋签约等。自营的“重”是目

前民宿团队无法承受的，经过衡量，前期采用平台模式，后期慢慢发展自营房东，以自主接入为主，最大限度减少线下操作。

在平台模式下，所有的同行都是朋友，团结一切可以团结的力量扩大在线房源数量。民宿团队先后接入了业内做得口碑比较好的几家供应商，迅速使携程民宿实现了从 0 到 1 的跨越。

在民宿人人都是主人翁

好的团队是整个业务顺利开展的坚实保障。我们在民宿团队使用不同于传统的管理模式，采用扁平化管理，并在团队成员中实现多角色轮换：



团队成员主动打破固有身份、角色、领域知识的思维限制，担任兼职客服，团队成员轮流值守，随身携带客服手机，及时跟进并处理用户下单。同时还兼职商务，与蚂蚁、途家等多家平台签署直连对接协议。另外，不定期电话回访并收集用户痛点，有针对性地提升用户体验及转化率。

为了更好地了解民宿业态以及更接近用户，在团队内部还进行了民宿房东样板房试验，多个团队成员将自己的房源提供出来，亲自体验担任民宿房东。

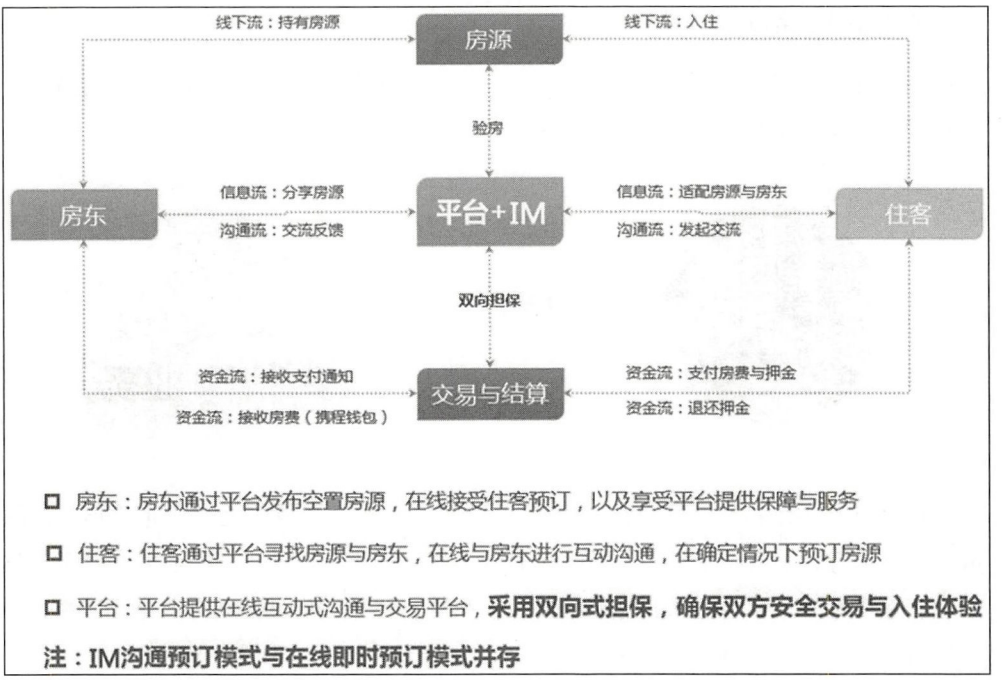
2. 马拉松式的小步快跑

在前一个阶段，已经完成了基本的流量和覆盖，这一阶段主要是聚焦目标，持续交付。在流量与覆盖持续增长下，对各方面进行综合优化，最小迭代周期到天，平均两天一个版本。

首先，把携程 APP 的民宿宫格资源进行整合，把民宿房源混排进宫格列表页，实现了 UV10 倍增长。另外，加强了与现有直连平台合作，洽谈了更多渠道，通过和途家、蚂蚁的合作，获取更多房源，同时新增了云掌柜、番茄来了等直连渠道。在商

户端进行优化，露出个人房东入口，通过代理增加民宿的房源入口。针对用户体验方面，优化了列表页排序，对详情页信息进行补充和完善，增加 IM 接入等，使用户下单流程更为便捷。与此同时，对运营后台功能的强化，自动化测试的覆盖率提高，有效地确保了需求的快速迭代，需求的上线率达到了 99%。

通过这一阶段的努力，实现流量和覆盖率增长了 6 倍，订单增长了 10 倍。



3. 资源重新整合，售卖渠道进一步扩大

民宿作为非标住宿，在展示以及售卖流程上和普通的酒店有所不同。通过建立标准化接口，可以将民宿资源分销到酒店频道售卖，一方面可以丰富酒店频道的产品资源，另一方面也大大提高了民宿产品的流量，提高了用户覆盖率。另外，海外也存在大量优质的民宿资源，将这些资源整合到民宿平台中，实现全球化的资源覆盖，给用户出行住宿提供极大的便利。

提前毕业

“非常高兴在金秋时节与大家分享一个好消息，我们与携程旅行网、去哪儿网达

成战略协议，并购携程旅行网、去哪儿网旗下公寓、民宿业务……”

2016 年 10 月 20 日途家宣布并购携程的民宿业务，携程民宿从携程内部一个创业孵化项目一下子成为了携程在非标住宿领域版图中的重要一分子，对标准类酒店形成了不可或缺的重要补充。

从 2014 年携程民宿正式开始起步，到 2016 年 10 月并入途家截止，短短 2 年多的时间，民宿团队通过各种创新的方式方法，在有限的资源下将民宿在携程内部做到了从无到有，从小到大。“恭喜你们提前毕业”这是梁建章在总结会上对整个民宿团队 2 年多来成绩的认可，也是民宿团队进一步争取更大成绩的激励。

一段征程的结束预示着另一段征程的开始，民宿团队在携程内部创业的征程上毕业了，但作为携程集团在非标住宿领域的重要一环，与其他合作伙伴一起共同服务好广大用户的道路上才刚刚开始。

6.3.4　小诗机，大创新

在 2016 年博鳌亚洲论坛的时候，携程创始人梁建章启动该项目，由技术团队及战略合作和营销创新部一起成立了人机诗歌项目组，并且盛邀顾文豪、朱钦运（茱萸）二位古典文学博士加入小诗机项目，使得计算机获得了专业的学术性指导。经历 7 个月的时间，分成 4 个不同阶段，完成了携程 AI 写诗水平从无到有，并且在深度学习、景点知识库、诗词语义框架体系、图片识别等多个领域取得了突破性进展。综合测评结果表明，携程 AI 的写诗质量和智能化已经远超过微软、百度、IBM、清华大学等竞品项目。

在 2017 年春节之际正式上线面向大众，引来了多位大佬关注。分众传媒董事长江南春赞不绝口；知名娱乐人高晓松也晒出了一首用携程 AI 写的命题诗《沙》，高晓松表示：“从这篇诗歌来看，已经颇有意境。估计再有两三年，就可以代替一部分没有特点只干行活儿的三四线作者。”

上线之初吸引了数百万人体验，写诗拜年成了一股新风潮，刷爆了朋友圈。数据统计显示，在春节期间，小诗机最高一天创作了近 10 万首诗，高峰时段平均一分钟要创作近 300 首诗。令人惊叹的是，一位女士 5 天竟作诗 60 余首，高小姐是北京一家外企女高管，2017 春节与闺蜜同行去巴厘岛度假，自从知道小诗机后，每到一个景

点便拍照作诗留念。她说，这比自拍有意思，玩起来上瘾，不知不觉就作了 60 多首诗，可能是转发朋友圈太多了，被朋友们笑称为“写诗狂魔”。

创新之路

小诗机之所以能够达到这样的效果，是一个持续创新、不断优化、不断迭代、不断进取的结果，总体来说，整个项目进行了下面几次大的阶段。

第一阶段　诗情画意 —— 饱读诗书，磕磕碰碰

该阶段的主要目标是一个从 0 到 1 的过程，简而言之，写出好的诗文，为了达到该目标整个项目分成以下几个步骤：

1．竞品分析及需求管理

诗歌有多种类型，包括律诗、绝句，五言、七言。整个团队研究了当时市场上的类似产品，如微软研究院、IBM 中国研究院的偶得、玻森科技的编诗姬等数十个产品，深挖每个产品的优缺点，最终确定该阶段我们的产品目标是七言律诗。

2．产品开发及技术创新

小诗机是一个基于大数据、机器学习等技术的产品。整个产品的开发包括下面的步骤：首先获取数据源，通过网络爬虫、人工运营等技术获取诗文，通过数据清洗，最终获取到现存全部的近 30 万首律诗作为数据源。其次提取数据特征及建立算法模型，基于第一步的数据源，深挖数据的特征，训练出对应的语言模型，并将古诗的上下句关系映射为统计翻译模型中的源语言和目标语言，构建统计机器翻译模型。最后成诗及优化，有了算法模型，使用遗传算法来对解空间寻求最优化，把古诗的生成看作是一个状态空间搜索问题。同时为了提供音律和押韵的正确性，接入遗传算法；为了满足用户的主题意向，除了基于清代刘文蔚的《诗学含英》，同时使用 word2vec、LDA、LSI 等方法构建和丰富主题词库。

第二阶段　游山玩水——千山万水，铭记于心

经过第一阶段，小诗机能够写出优秀的诗歌了，下面如何结合公司的业务特点呢？考虑到携程是一家旅游公司，积累了丰富的景点特征、景点照片以及用户的评论，然后确定了第二阶段的主要目标是能基于景点进行写诗，那么该阶段的主要开发和创新在于建立全球景点知识库，分成下面几个步骤。

首先获取数据源，除了公司已有的景点简介、用户评论等数据，为了丰富数据，弥补部分景点数据的稀疏性，还结合了维基百科、百度百科为代表的大规模知识库。

其次提取数据特征及建立算法模型，对公司已有的景点简介、用户评论等大量的文本数据进行深入的清洗，以及对关系、属性提取进行信息挖掘，获得每个景点基本的信息，如景点特色、周边景点，构建较为全面的知识图谱，并给每个特征赋予不同的权重，然后通过景点的特征进行汇总和聚合，形成各个城市的特色及权重。同时维基百科、百度百科等外部数据包含了大量结构化的知识，可以高效地应用到知识图谱。如何合理地结合这两部分数据，消除两部分数据的复杂性和差异性，我们使用同义词自动获取、词义消歧等方法进行关系和属性的进一步抽取和融合。同时基于不同数据源的置信度、不同属性关系的统计分布进行重要性评估，使得对现实景点等信息之间的分布有了更为精确的描述。

第三阶段　多愁善感——寓情于景，升华情境

有了景点，有了诗文，通过对比测试发现，小诗机的输出少了些诗人的多情善感，所以开始了第三阶段的技术创新，让小诗机的输出更加人性化、生动、富有逻辑。整个阶段的产品开发和技术创新包括：

首先诗文流利性，一方面对写景诗歌的成诗体系进行分析和挖掘，提取常用语义，并构建三层语义体系结构，从而能从多个层次把握诗歌的语义搭配粒度。另一方面基于提取出的语义结构，挖掘常用诗歌语义模型，将诗歌语义模型和语义层级相结合，使得表达更加流畅。

其次建立情感知识库，基于古诗词语料以及一些现代文语料进行文本的情感分析、挖掘、归纳和推理，获取不同情感下的表达方式，构建情感词的近义词集合，以及相关的情感知识库。同时构造情感引擎，能够根据当前不同的情感，给候选词渲染不同的情感色彩，为诗整体渲染不同的情感基调。

最后成诗算法优化，一方面基于当前景点和对应主题，从主题、平仄、韵律、对仗、场景等方面进一步优化写诗算法，从而保证诗的韵味、主题的凸显、语义的连贯。另一方面，构建逻辑引擎，保证生成的诗中，不会出现矛盾的场景，如季节引擎、时间引擎、景点引擎等。

第四阶段　看图作诗——图片识别，趣味生动

经过前三阶段的发展，整个小诗机已经具备了面向大众的能力，如何让小诗机变得有娱乐性，又能包含公司的基因呢？考虑到携程的用户群体的特征、大众的自嗨心理，产品的简单性，确定最终输出的样式是用户上传图片、小诗机写诗、大家再分享。这个阶段的产品开发和技术创新包括：

首先图像识别技术创新，小诗机采用 CNN 深度神经网络，能够识别的景物、人物类别超过 1500 种，包括人物的性别、年龄及神情等。

其次诗文输出的完善，基于图像识别技术，自动获取图片中的景点、物品、人物等信息，同时结合图片等 GPS 信息，获取当前的景点信息。有了用户行为情感和景点，整个诗歌的输出做到了寓情于景。看图作诗如下图所示。

后　记

当作者团队找我来为本书做一个后记的时候，我本来是拒绝的。因为按照最近“中国首位 00 后 CEO”的说法，作为三四十岁的老一辈从业人员，我可能已经没办法再了解互联网，更不用说出来评头论足了。但面对他们殷切的眼神，我勉强厚着老脸应承下来，忐忑着是否还能胡诌上几句。

先粗粗地浏览了目录，再细细地读了一大半的章节。一个个颇具专业深度，但又不失生动有趣的案例跃然纸上。本书竟然完全出乎了我的意料！多年前，我亲手选拔并组建了携程技术体系的第一支项目经理团队，后续又作为产品经理，或亲身经历，或有幸见证了许许多多项目和产品历尽千辛万苦的脱颖而出。我知道这离不开一套套既有行业高度，又十分接地气的适合携程的方法论。而本书通过一个个鲜活的故事，恰到好处地阐述了这些方法论，及其运用之妙，在乎一心。这对于后来的携程人，以及其他关心这个领域的读者，都颇具参考意义。

本书的许多作者都是我的亲密战友。他们平日战斗在商战的第一线，还有心利用闲暇时间提炼和记录下这些项目和产品的心得，这是非常难能可贵的。虽然本书的语言可能不及职业作家优美，专业可能不及研究机构严谨，但字里行间，我看到了他们的热情、诚意和自信。衷心期待各位读者也能和我一样收获良多。

度假事业部 COO/攻略社区事业部 CEO　喻晓江

致　谢

感谢携程集团 CTO 甘泉的大力支持，尤其感谢特邀撰稿人贡献内容，本书才得以问世。

本书大部分内容都是在周末和晚上完成的，谢谢本书各章节出品人的辛苦付出。同时感谢敏捷交流群的小伙伴们为本书锦上添花，谢谢机票产品部的摄影大师邱冰清为本书作者团队拍照。

最后还要感谢电子工业出版社为我们出版本书，特别感谢编辑孔祥飞，帮助我们去芜存菁。谢谢你们的支持。

特邀撰稿

陈刚——携程集团高级副总裁/火车票事业部 CEO/创新工场 CEO

OK 制基本运营思路 / 130

首先要有个大目标 / 132

李巍——携程酒店高级研发总监

组建小团队，你可以这么玩/51

徐鑫——携程酒店研发团队 PO

如何成为一名优秀的 Product Owner / 42

王玉琛——携程集团副总裁/车船票事业部 CEO/租车事业部 CEO

数学分析决策模型在 MTP 和 OKR 的应用 / 133

罗昭君——携程机票无线测试总监
敏捷是什么/ 3
速度快——自动化让团队飞 / 65

俞鋆——携程 Scrum Master
从团队成员的视角看敏捷 / 86
持续优化——让团队更出色 / 109

尹谜眉——携程产品经理
与团队共成长——产品新兵成长记 / 88

彭微——携程高级产品经理
如何培养团队目标感——让团队更优秀 / 107

钟良敏——携程用车业务前产品总监
OKR 与 Project / 136

彭廷——携程租车业务产品总监
OKR 在租车团队的实践 / 137

钟磊——携程租车业务技术总监
一个“演员”的自我修养 / 138

许方佩——携程汽车票高级产品经理
OK 制是“两个人”的事，OKR 是“一群人”的事 / 141

谈寅——携程火车票高级研发经理
完成不可能目标的可行方案 / 142

王晨——携程汽车票研发经理
OKR 在汽车票数据系统的实践 / 144

吴其敏——携程框架高级研发总监
王兴朝——携程框架研发总监
高峻——携程运维研发总监
王潇俊——携程系统研发高级总监
陈逸——携程系统研发技术专家
陈劼——携程应用运维高级总监
后台应用持续交付实践 / 160

刘李丰——携程基础研发高级经理
一步一步迭代出无线持续交付平台/ 177

苏玲——携程系统研发高级经理
周光明——携程系统研发资深工程师
持续集成，让团队没有难集成的代码 / 149

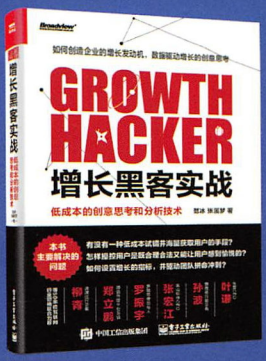
徐豪杰——携程运维研发经理
运维工作流平台的演进之路/187

宋培培——去哪儿 高级项目经理
打造一站式项目管理平台，助力研发效率提升 / 216

孙磊——携程度假运营总监
自上而下的产品，如何快速实施做大 / 230

时宝丽——携程度假运营总监
打配合战的产品，如何快速上线做大 / 233

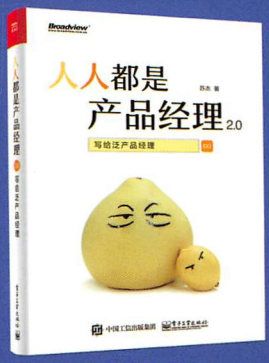
好书力荐



《增长黑客实战》

给创业型公司如何搭建增长团队的实践指南；吴晓波、李笑来、任富佳、戴雨森、楼军联合力荐！

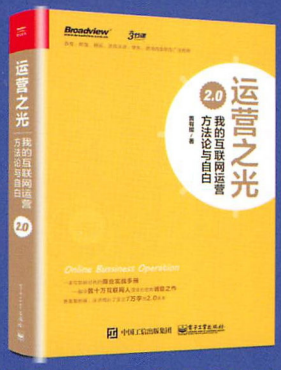
ISBN：978-7-121-31410-0



《人人都是产品经理 2.0——写给泛产品经理》

人人家族再添新丁，双剑合璧纵横职场，格局扩至全互联网，深度覆盖创业人群！

ISBN：978-7-121-31140-6



《运营之光 2.0：我的互联网运营方法论与自白》

百度阿里腾讯美团点评，滴滴内部广泛传阅的运营力作，豆瓣“十大商业经管类好书”！

ISBN：978-7-121-31154-3

读完本书后，如果对您有用，您就给我们点个赞。我们的故事都是真实的，通过这些案例，希望让您感受到组织的变革、管理方法和工具的变革是多么不易，这使我们受益良多。成长中难免会犯错，但 18 岁的携程永远不会停止探索。谨以此文与各位一起共勉！

大咖力荐

实践、学习、思考、再实践，携程的管理实践还在路上，希望能够帮助读者，携手成就精彩旅程。

—— 携程集团首席运营官 孙茂华

公司的经营就是要做到物尽其用，人尽其才。这是一本充满实战的项目管理实践经典。如何让21世纪宝贵的人才充分地发挥自己的能量，全书内容是这方面全面实践的记录。

—— 去哪儿网首席执行官 谌振宇

本书无疑是创业公司、正在敏捷转型公司的福音，书中的很多方法可以直接拿来参考和使用，使您少走很多弯路。

—— 途家首席运营官 杨昌乐

通过本书可以看到携程自上而下的敏捷转型实战，以及扎实的细致落地案例。尤其是独具匠心的团队和产品管理，以及曝光的产品研发交付细节，都瞄准了互联网产品研发过程中的痛点，成功必有其道！

—— 网易杭研项目管理部总监 曹智清

本书介绍了一个个相对独立的以价值交付为目标的价值单元团队的作战方式，有助于团队各角色聚焦在共同的业务目标上，在一起高效协同地专注在某个业务方向，有归属感、价值感、成就感，快速灵活地响应业务变化，实现业务价值交付，值得大多数企业借鉴。

—— 百度资深敏捷教练 姜丽芬 Jennifer

这是一个变化越来越快的社会，有幸从事产品创新的工作，我深知应对未知和变化的方法就来自敏捷的思想内核。这本携程团队带来的新书，既有理论高度，又有实操手法，感谢你们。

——《人人都是产品经理》作者 苏杰

互联网公司敏捷转型真实案例总结，内容翔实，鲜活生动，值得借鉴参考。

—— 京东首席敏捷创新教练 王立杰

敏捷做到知行合一不易，国内的敏捷现状是知多行少，独到的敏捷实践少之又少。本书精选了携程的敏捷实践，可谓独到。

—— 银联支付学院高级经理 于兆鹏

博文视点Broadview

新浪微博 weibo.com
@博文视点Broadview

策划编辑：孔祥飞
责任编辑：徐津平
封面设计：李　玲　吴海燕

上架建议：互联网运营

ISBN 978-7-121-32903-6

9 787121 329036 >

定价：59.00元